НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Обчислювальної техніки

До захисту допущено:
Завідувач кафедри
Сергій СТІРЕНКО_____
«___»_____2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»
на тему: «Системи автоматизації збірки та тестування розгортування додатків»

Виконав:
студент IV курсу, групи ІО-61з
Карабчук Володимир Русланович

_____

Керівник:
Доцент кафедри ОТ, к.т.н.,
Павлов Валерій Генадійович

_____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.,
Сімоненко Валерій Павлович

_____

Рецензент:
Професор кафедри АУТС, д.т.н.,
Корнієнко Богдан Ярославович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломного проєкту

освітньо-кваліфікаційного рівня__бакалавр___

на тему: “ Системи автоматизації збірки та тестування розгортування додатків ”

Київ – 2020 року

Технічне завдання на розробку системи автоматизації збірки тестування та розгортування додатків з використанням технології тестового розгортування Blue/Green.

ВСТУП

Розробити модель системи автоматизації збірки тестування та розгортування додатків з використанням технології тестового розгортування Blue/Green. Модель передбачає подальший розвиток в стандартизований процес розробки, призначену для всіх видів автоматизації веб-додатків.

Підстави для розробки

Підставами для розробки є можливість зменшити вартість та збільшити якість програмного продукту, за рахунок використання системи автоматизації.

Призначення розробки

Модель є реалізацією складного комплексу систем автоматизації збірки тестування та розгортування додатків, призначеного для впровадження і використання в поточних розробках. Призначення системи - реалізувати новий підхід до систем атоматизації, що дозволяє розробникам мати можливість покращити якість свого програмного продукту, а замовникам зменшити витрати на його створення та підтримку.

Технічні вимоги для апаратної частини комп’ютера

- Комп’ютер під операційною системою Linux;
- Процесор з 2 ядрами тактовою частотою не менше ніж 2 ГГц;
- Оперативна пам’ять об’ємом не менше ніж 4 Гб;
- Будь-який браузер з підтримкою javascript

Вимоги до функціональних характеристик кінцевого продукту.

- Бути зручним у використанні
- Бути вигіднішим, ніж купівля серверів
- Мати низький поріг входу для розробників
- Найкраще співвідношення ціна-якість
- Мінімізувати споживання ресурсів апаратної частини
- Підвищувати якість розробки
- Мати можливість швидко відновити роботоспроможність веб додатку в разі помилок
- Надійність системи

Етапи розробки

№	Зміст роботи	Термін	Виконавець
1	Огляд літератури та інших джерел з інформацією про системи автоматизації збірки та розгортування додатків	01.09.2019	Карабчук В.Р
2	Формулювання проблеми для вирішення	01.01.2019	Карабчук В.Р
3	Аналіз існуючих рішень	28.02.2020	Карабчук В.Р
4	Огляд переваг та недоліків існуючих рішень	30.02.2020	Карабчук В.Р
5	Формування власного варіанту вирішення задачі	20.03.2020	Карабчук В.Р
6	Формування переліку питань, що потребують детального огляду	27.03.2020	Карабчук В.Р
7	Розробка системи автоматизації збірки та тестування додатків	25.04.2020	Карабчук В.Р
8	Розробка системи розгортування на основі технології Blue/Green	22.05.2020	Карабчук В.Р
9	Об'єднання розроблених частин в єдину модель.	26.05.2020	Карабчук В.Р
10	Здача і захист курсового проекту.		Карабчук В.Р

Аннотація

До бакалаврської дипломної роботи Карабчука Володимира на тему:

«Система автоматизації збірки та тестування розгортування додатків»

Дипломна робота присвячена вирішенню проблеми автоматизації збірки та тестування розгортування додатків

Запропонований підхід дозволить виконувати наступні задачі: автоматизації збірки, тестуванная, реалізацію тестування розгортування за допомогою системи розгортування Blue/Green, реалізацію економічних рішень на базі хмарних технологій. Після того як розробник відправляє свій оновлений код до системи автоматизації, вона автоматично збирає код, виконує автоматичні тести написані тестувальниками, формує Docker зображення, у разі успішного завершення всіх попередніх етапів та відправляє до сховища контейнерів. Оператор створює нову задачу для хмари з використанням нового зображення і контролює етап розгортування поступово, переключаючи трафік на новий контейнер. В разі похибок оператор може в будь момент відмінити розгортування, просто переключивши трафік на попередній контейнер.

Аннотация

К бакалаврской дипломной работе Карабчука Владимира на тему:

«Система автоматизации сборки и тестирования развертывания приложений»

Дипломная работа посвящена решению проблемы автоматизации сборки и тестирования розвертывания приложений.

Предложенный подход позволит выполнять следующие задачи: автоматизации сборки, тестирования, реализацию тестирования розвертывания с помощью системы розвертывания Blue / Green, реализацию экономичних решений на базе облачных технологий. После того как разработчик отправляет свой обновленный код к системе автоматизации, она

автоматически собирает код, выполняет автоматические тесты написаны тестировщиками, формирует Docker изображения, в случае успешного завершения всех предыдущих этапов, и отправляет в хранилище контейнеров. Оператор создает новую задачу для облака с использованием нового изображения и контролирует этап розвертывания постепенно, переключая трафик на новый контейнер. В случае ошибок, оператор может в любой момент отменить розвертывания, просто переключив трафик на предыдущий контейнер.

Abstract

To the bachelor's thesis of Vladimir Karabchuk on the topic:

"Automation system for assembling and testing rozgortuvannya applications"

The thesis is dedicated to solving the problem of automation of assembly and testing deployment of applications.

The proposed approach will allow you to perform the following tasks: automation of assembly, testing, implementation of deployment testing using the Blue / Green deployment system, implementation of cost-effective solutions based on cloud technologies. After the developer sends his updated code to the automation system, it automatically collects the code, performs automatic tests written by testers, generates Docker images, in case of successful completion of all the previous steps, and sends it to the container storage. The operator creates a new task for the cloud using a new image and controls the deployment phase by gradually switching traffic to a new container. In case of errors, the operator can cancel the deployment at any time simply by switching traffic to the previous container.

Перелік скорочень

AWS - Amazon Web Servises (хмара, яка належить Amazon)

GCP - Google Cloud Platform(хмара, яка належить Google)

CI - Continious Integration(методологія неперервного оновлення коду)

CD - Continious Delivery(методологія розгортування нових проектів)

API - Application programming interface(програмний інтерфейс)

GKE - Google Kubernetes Engine(сервіс для управління Kubernetes кластером)

ОС – Операційна система

ЗМІСТ

ВСТУП

Зі збільшенням кількості додатків та їх цінності, стає актуальнішою проблема забезпечення розробників та тестувальників автоматичною системою, що зможе збирати всі автоматичні тести та код і автоматично проводи всі механічні операції по збірці та тестуванню. Збільшення кількості проектів, що розроблюються ставить жорсткі рамки по швидкості розробки. Більш того, вартість послуг розробників стає ще вищою, тому потрібно знайти спосіб зменшити час розробки, тим самим зменшити її вартість.

З вищеописаного висувається завдання: вирішити проблему забезпечення розробників надійного та швидкого способу, автоматизувати всі можливі етапи розробки . Для формування ефективних кроків та для виконання цього завдання потрібно: виділити характеристики, ознаки та особливості сьогоденного стану систем автоматизації, проаналізувати ефективність та дієвість методів та засобів, що існують наразі, щодо оптимізації навантаження та автоматизації на ресурси що використовуються в розробці.

Метою дипломного проекту є:

Знайти найбільш оптимальний спосіб автоматизувати процес збірки та тестування додатків з подальшим розгортуванням на базі технології тестового розгортування

Для досягнення цієї мети необхідно вирішити наступні задачі:

1. Знайти спосіб автоматизувати процесс збірки та тестування
2. Проаналізувати і порівняти існуючі варіанти автоматизації
3. Проаналізувати і порівняти існуючі варіанти віртуалізації
4. Проаналізувати і порівняти існуючі варіанти хмарних обчислень
5. Проаналізувати існуючий підхід до тестування розгортування
6. Обрати найбільш оптимальну збірку технологій з автоматизації, віртуалізації, хмарних обчислень з технологією тестового розгортання

РОЗДІЛ 1
ОГЛЯД МЕТОДІВ АВТОМАТИЗАЦІЇ ТА ТЕСТУВАННЯ ДОДАТКІВ

1.1 Аналіз проблеми автоматизації існуючих етапів розробки

Наразі, в деяких існуючих проектах, використовуються процеси збірки та тестування використовуючи для цього спеціаліста, який б просто прописував потрібні команди в потрібний час, передавав деякі параметри необхідні для збірки та тестування проекту перед тим як можна буде використовувати його, як новий реліз нашої програми. Це сприяє тому, що вам необхідно платити спеціалісту роботу, якого можна оптимізувати, більш того, так як всі ці речі робить людина, не виключено що вона може помилитися і тоді в реліз піде не робоча програма, яка може зупинити роботу вашого додатку на довгі години. Найчастіше цю проблему вирішують написанням скрипту, який би автоматизував всі необхідні дії, потрібні для перевірки вашого коду, але це може спрацювати, якщо у вас не велика команда і ви робите оновлення не частіше одного разу у неділю.

Уявимо, що відбудеться якщо у нас 100 розробників, що поділені на 4 команди і кожна з команд хоче проводити оновлення кожен день або по декілька разів на день, а ще, якщо наш додаток розміром в 4 гігабайти - це може бути десятки збірок, що займають годину або більше на весь процес збірки та тестування, тому нам потрібна система розподіленої потужності, що могла б запускатись на великій кількості комп’ютерів, проте використовувати єдиний центр управління. Це створює іншу проблему. Дуже важко використовувати єдиний реєстр бібліотек, які використовуються на комп’ютерах в розподіленій системі та бути повністю впевненими в тому, що програма не залишає кеш, який потім буде використовуватися для збірки додатку. Насправді, проблема кешу доволі актуальна. Кеш може приховати існуючу проблему на тривалий час, і на пошук та вирішення цієї проблеми може піти величезна кількість часу.

З вище сказаного, можна зробити висновок, що потрібна система що змогла б забезпечити проект алгоритмами автоматизації, розподіленими потужностями, гарантувала б однаковість всіх оточень в нашій системі, розподілених потужностей і при кожній збірці створювала б нове віртуальне оточення.

Для систем, що можуть робити вище сказане є навіть назва CI або безперервна інтеграція (Continuous integration)

1.1.2 Огляд наявних засобів систем автоматизації з можливостями розподіленої потужності

Найбільш популярні программи для реалізації **CI** процессів - це **Jenkins**, **Gitlab**, **Travis CI** та **CircleCi**. В кожної є свої переваги, їх ми і розглянемо далі, а також порівняємо їх.

CircleCI

CircleCI - це хмарна система, для якої не потрібно налаштовувати окремий сервер і яку не доведеться адмініструвати. Однак існує і локальна версія, яку ви можете розгорнути в приватному хмарі.

Навіть для комерційного використання існує безкоштовна версія.
За допомогою REST API можна отримати доступ до проектів, збірок і артефактів. Результатом збірки є артефакт або група артефактів. Артефактом можуть бути скомпільований додаток або виконувані файли (наприклад, APK для Android), або метадані (наприклад, інформація про вдало завершився тестуванні).**CircleCI** кеширує сторонні залежності, що дозволяє уникнути постійного встановлення необхідних оточень.

Існує можливість підключення до контейнера по SSH. Це може знадобитися, якщо виникнуть якісь проблеми.

CircleCI - повністю готове рішення, яке потребує мінімального налаштування.

CircleCI сумісна з:

- Python, Node.js, Ruby, Java, Go і т. Д .;
- Ubuntu (12.04, 14.04), Mac OS X (платні акаунти);
- Github, Bitbucket;
- AWS, Azure, Heroku, Docker, виділений сервер;
- Jira, HipChat, Slack.

Переваги CircleCI:

- легкий та швидкий початок роботи;
- безкоштовна версія для комерційного використання;
- невеликі та легко читаються файли конфігурації в форматі YAML;
- відсутність необхідності у виділеному сервері CircleCI.

Недоліки CircleCI:

- CircleCI в безкоштовній версії підтримує лише Ubuntu 12.04 і 14.04.
- Для використання MacOS доведеться заплатити.

Незважаючи на те, що **CircleCI** може працювати з будь-якими мовами програмування, з коробки підтримуються тільки Go (Golang), Haskell, Java, PHP, Python, Ruby / Rails, Scala.

При бажанні налаштувати систему під себе, в деяких випадках можуть виникнути проблеми, і тоді для досягнення мети знадобиться додаткове програмне забезпечення.

Також, візьмемо до уваги, що у хмарних систем є безсумнівні переваги. Потрібно бути готовим до того, що в будь-який момент необхідна вам функція може бути видалена, і нічого з цим вдіяти не можна.

Travis CI

Travis CI і CircleCI дуже схожі.

Обидві системи:

- використовують файли конфігурації в форматі YAML;
- розгорнуті в хмарі;
- підтримують Docker для запуску тестів.

Що є в TravisCI і немає в CircleCI?

- Запуск тестів одночасно під Linux і Mac OS X.
- Підтримка більшої кількості мов «з коробки»:
- Android, C, C #, C ++, Clojure, Crystal, D, Dart, Erlang, Elixir, F #, Go, Groovy, Haskell, Haxe, Java, JavaScript (with Node.js), Julia, Objective-C, Perl, Perl6, PHP, Python, R, Ruby, Rust, Scala, Smalltalk, Visual Basic.
- Підтримка build matrix.

Build matrix - це інструмент, який дає можливість виконувати тести, використовуючи різні версії мов і пакетів. Він володіє багатими можливостями по налаштуванню. Наприклад, при невдалих збірках в деяких середовищах, система може видати попередження, але збірка цілком не буде вважатися невдалою (це зручно при використанні dev-версій пакетів).

Переваги Travis CI:

- build matrix «з коробки»;
- швидкий старт;
- невеликі і легко читаються файли конфігурації в форматі YAML;
- безкоштовна версія для opensource-проектів;
- відсутність необхідності у виділеному сервері.

Недоліки Travis CI:

- в порівнянні з CircleCI ціни вище, немає безкоштовної версії для комерційного використання;
- обмежені можливості по налаштуванню (для деяких речей може знадобитися сторонній софт).

Jenkins

Jenkins або **дженкінс** - це інструмент автоматизації з відкритим кодом, написаний на Java, з плагінами, побудованими для безперервної інтеграції. Jenkins використовується для того, щоб постійно розробляти та тестувати програмні проекти, що полегшує розробникам інтеграцію змін у проекті та полегшує користувачам отримання свіжого релізу. Це також дозволяє безперервно поставляти програмне забезпечення, шляхом інтеграції з великою кількістю технологій тестування та розгортування.

Jenkins був розроблений у 2004 році компанією Sun Microsystems. В 2010 році Sun Microsystems була купленою компанією Oracle, проте через проблеми з іменними правами, Jenkins став програмою з відкритим вихідним кодом та підтримується спільнотою розробників.

З Jenkins організації можуть прискорити процес розробки програмного забезпечення за допомогою автоматизації. Цей інструмент інтегрує процеси життєвого циклу розвитку всіх видів, включаючи складання, документування, тест, пакет, етап, розгортування, статичний аналіз та багато іншого.

Jenkins досягає постійної інтеграції за допомогою плагінів. Плагіни дозволяють інтегрувати різні етапи DevOps. Для інтегрування певних інструментів, потрібно встановити додатки для цього інструменту. Наприклад: Git, проект Maven 2, Amazon EC2, видавець HTML тощо.

До переваг Jenkins належать:

- Це інструмент з відкритим кодом та великою підтримкою громади.
- Це легко встановити.
- Він має 1000+ плагінів, щоб полегшити вашу роботу. Якщо плагіна не існує, ви можете його кодувати і поділитися зі спільнотою.
- Це безкоштовно.
- Він побудований за допомогою Java і, отже, є портативним для всіх основних платформ.
- Має систему розподіленої потужності на основі Jenkins агентів

GitLab

GitLab - це система управління сховищами Git. Він написаний на Ruby та дозволяє легко розгорнути змістовний контроль версій для коду. Вперше він був опублікований на GitHub у жовтні 2011 року, і з того часу перетворився на потужний інструмент. GitLab публікується під ліцензією MIT, але при перерозподілі коду обов'язково слід згадувати про автора. GitLab має багато можливостей. Напевно, найвидатнішими - є зручний веб-інтерфейс та можливості управління дозволами. Вбудовані функції для документації проектів або відстеження необхідних змін, за допомогою трекера випусків- це дуже вагомі моменти для GitLab.

Веб-інтерфейси

Веб-інтерфейс у багатьох випадках нагадує той, який використовується у GitHub, що виявилося дуже функціональним для багатьох розробників. Це дуже послідовно і просто, незважаючи на кількість функцій, які він містить. Можна відкрити запити на об'єднання для гілок, переглядати різні або навіть редагувати файли прямо у веб-інтерфейсі, а потім поетапно змінювати зміни на коміт.

Управління дозволами

У GitLab є дуже добре розроблені ролі, що дозволяє застосовувати їх. Від гостя до майстра, у вас є різноманітні можливості обмежити та надати

доступ до функцій, таких як переміщення до сховищ або до тих гілок, на які розробник може натиснути, щоб була можливість контролювати, який код потрапляє у продукт чи проект.

Документування вашого проекту

Документування проектів може бути важкою справою, оскільки його часто не описують, як важливе завдання з самого початку. GitLab має чудові інструменти, щоб зробити їх більш значущими та швидшими у створенні. За допомогою вбудованого відстеження випусків та вікі-файлів, ви можете перетинати довідкові файли, коміти, та звичайно, інші вікі-сторінки. Це чудові інструменти, які дозволяють команді утримувати всіх на одній сторінці та запобігати перетворенню проектів, незалежно від складності, у щось важке для розуміння.

GitLab CI

Щоб використовувати GitLab CI все, що потрібно - це база коду додатків, розміщена в сховищі Git, а для сценаріїв побудови, тестування та розгортування слід вказати у файлі під назвою .gitlab-ci.yml, що знаходиться в кореневому шляху вашого сховища.

У цьому файлі можна визначити сценарії, які потрібно запустити, відзначити включення та кешування залежностей, вибрати команди, які потрібно запустити послідовно, і ті, які потрібно запустити паралельно, визначити, де ви хочете розгорнути додаток, і вказати, чи захоче запустити сценарії автоматично або запустити будь-який з них вручну.
Ознайомившись з GitLab CI , можна додати більш досконалі кроки у файл конфігурації.

Щоб додати сценарії до цього файлу, вам потрібно буде впорядкувати їх у послідовності, яка відповідає потрібній програмі та відповідає тестам, які тестувальники захочуть виконати. Щоб візуалізувати процес, уявіть, що всі

сценарії, які додаються у файл конфігурації, збігаються з командами, які виконуються на терміналі на вашому комп’ютері.

Щойно буде доданий файл конфігурації .gitlab-ci.yml у сховище, GitLab виявить його та запустить сценарії за допомогою інструменту, під назвою GitLab Runner, який працює аналогічно терміналу.

Сценарії згруповані в завдання, і разом вони складають конвеєр.

До переваг GitLab CI належать:

- Це інструмент з відкритим кодом.
- Це легко встановити.
- Це безкоштовно.
- Вбудований Github
- Gitlub runner для реалізації розподілених навантажень
- Високий рівень безпеки

1.1.3 Порівняння Gitlab, Jenkins, Travis ci і CircleCi

Табл 1.1 Порівняльна таблиця

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Вбудований CI GitLab і Jenkins мають вбудовану безперервну інтеграцію безкоштовно, не потрібно встановлювати її окремо. Використовуйте його для створення, тестування та розгортування вашого веб-сайту . Результати роботи відображаються на запитах та об'єднання для легкого доступу.	+	+	+	+
Моніторинг продуктивності додатків GitLab збирає та відображає показники продуктивності, для розгорнутих додатків, використовуючи Prometheus. Розробники можуть визначити вплив злиття та стежити за своїми виробничими системами, не залишаючи GitLab.	-	+	-	-
Простота розгортування	-	+	+	+
Значення потокової аналітики GitLab пропонує інформаційну панель, яка дозволяє командам вимірювати час, необхідний для переходу від планування до моніторингу. GitLab може надати ці дані, оскільки в ньому вбудовані всі інструменти: CI, від перегляду коду до розгортування.	-	+	-	-

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Вбудований реєстр контейнерів Реєстр контейнерів GitLab - це безпечний і приватний реєстр для зображень Docker. Це дозволяє легко завантажувати зображення з GitLab CI. Він повністю інтегрований з керуванням сховищами Git.	-	+	-	+
Попередній огляд ваших змін за допомого программи Review Apps За допомогою GitLab CI ви можете створити нове середовище для кожної своєї галузі, прискоривши процес розвитку. Розгорніть динамічне середовище для ваших запитів, на об'єднання з можливістю попереднього перегляду філії в реальному середовищі. Програми для перегляду підтримують статичну та динамічну URL-адреси.	-	+	-	-
Комплексний API GitLab надає API для більшості функцій, що дозволяє розробникам створювати більш глибокі інтеграції з продуктом.	+	+	+	+
Горизонтальне автоматичне масштабування Натурна архітектура хмарних технологій GitLab CI може легко масштабувати горизонтально, додаючи нові вузли,				

Опис	Jenkins	Gitlab	Travis ci	CircleCi
якщо збільшується навантаження. GitLab Runners може автоматично пересувати нові контейнери, щоб забезпечити негайну обробку трубопроводів та мінімізацію витрат.	-	+	-	+
Приладова панель трубопроводів Візуалізуйте історію, поточний стан трубопроводів у проектах та групах на єдиній інформаційній панелі, яку можна налаштувати для кожного користувача.	+	+	+	+
Створений для використання контейнерів та Docker GitLab постачається з власним реєстром контейнерів, Docker CI Runner, і готовий до повного робочого процесу з контейнерами CI. Не потрібно встановлювати, налаштовувати або підтримувати додаткові плагіни.	-	+	-	-
Вбудований клауд GitLab та його CI - це Cloud Native, створений для хмарної моделі. GitLab можна легко розгорнути на Kubernetes і використовувати для розгортування вашої програми в Kubernetes Clusters.	-	+	-	+

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Налагодження контейнерів з інтегрованим веб-терміналом Легко налагоджуйте свої контейнери в будь-якому оточенні за допомогою вбудованого веб-терміналу GitLab. GitLab може відкрити термінальний сеанс безпосередньо з вашого оточення, якщо ваша програма розгорнута в Kubernetes. Це дуже потужна функція, де ви можете швидко налагоджувати проблеми, не залишаючи зручності веб-браузера.	-	+	+	+
Графіки трубопроводів Трубопроводи можуть бути складними конструкціями з безліччю послідовних і паралельних завдань. Щоб трохи легше побачити, що відбувається, ви можете переглянути графік одного трубопроводу та його стан.	+	+	+	+
Артефакти, що проглядаються За допомогою GitLab CI ви можете завантажувати артефакти вашої роботи у сам GitLab, без необхідності зовнішньої служби. Через це, артефакти також можна переглядати через веб-інтерфейс GitLab.	-	+	-	-

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Плановий пуск трубопроводів Ви можете змусити ваші трубопроводи працювати за розкладом у схожих на крон умовах.	+	+	+	+
Якість коду Звіти про якість коду доступні в області віджетів, запиту на злиття або на сторінці конвеєра, що дає вам раннє уявлення про те, як зміни вплинуть на здоров'я вашого коду, перш ніж вирішити, чи бажаєте ви його прийняти.	-	+	-	+
Багатопроектні конвеєрні графіки За допомогою графіків багатопроектного конвеєра ви можете бачити: як трубопроводи пов'язані між собою для проектів, які пов'язані з іншими через тригери.	-	+	-	-
Змінні для середовища змін Обмежте область змінної середовища, визначивши, для яких середовищ вона може бути доступна.	-	+	-	+
Захищені змінні Ви можете позначити змінну як "захищену", щоб вона була доступною лише для завдань, що працюють на захищених гілках, тому доступ до неї можуть отримати лише авторизовані користувачі.	-	+	-	-

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Середовища та розгортування GitLab CI здатний не тільки тестувати або будувати ваші проекти, але і розгортати їх у вашій інфраструктурі, з додатковою перевагою, що дає вам спосіб відстежувати розгортування. Середовища - це теги для ваших CI завдань, описуючи, де код розгортається.	-	+	-	-
Історія змінних Історія середовищ дозволяє вам бачити, що зараз розгорнуто на ваших серверах, і отримати доступ до детального перегляду для всіх минулих розгортань. З цього списку ви також можете повторно розгорнути поточну версію або навіть відкатати стару стабільну, якщо щось пішло не так.	-	+	-	-
Змінні на рівні групи Визначте змінні на рівні групи та використовуйте їх у будь-якому проекті групи.	-	+	-	-
Настроюваний шлях для конфігурації CI Ви можете визначити спеціальний шлях до вашого сховища для файлу конфігурації CI.	-	+	+	+

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Запустіть завдання CI в Windows GitLab Runner підтримує Windows і може виконувати завдання на цій платформі. Ви можете автоматично створювати, тестувати та розгортати проекти на базі Windows, використовуючи PowerShell або пакетні файли.	+	+	+	+
Запустити завдання CI на MacOS GitLab Runner підтримує MacOS і може виконувати завдання на цій платформі. Ви можете автоматично створювати, тестувати та розгортати для проектів на основі MacOS, використовуючи сценарії оболонки та інструменти командного рядка.	+	+	+	+
Запустіть завдання CI на Linux ARM GitLab Runner підтримує операційні системи Linux в архітектурах ARM і може виконувати завдання на цій платформі. Ви можете автоматично створювати, тестувати та розгортати для Linux на базі проектів ARM, використовуючи сценарії оболонки та інструменти командного рядка.	+	+	+	+

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Запустіть завдання CI на FreeBSD GitLab Runner підтримує FreeBSD і може виконувати завдання на цій платформі. Ви можете автоматично створювати, тестувати та розгортати проекти на базі FreeBSD, використовуючи скрипти оболонки та інструменти командного рядка.	+	+	+	+
Показати швидкість покриття коду для ваших трубопроводів GitLab здатний аналізувати журнали вихідних завдань та здійснювати пошук за допомогою настроюваного регексу будь-якої інформації, створеної за допомогою таких інструментів, як SimpleCov, щоб отримати охоплення кодом. Дані автоматично доступні в інтерфейсі користувача, а також як знак, який ви можете вставляти на будь-яку сторінку HTML або публікувати за допомогою GitLab Pages.	+	+	+	+
Детальніше про тривалість кожного виконання команди в GitLab CI Інші системи CI показують час виконання для кожної команди, виконаної у завданнях CI, а не лише загальний час. Ми переглядаємо, як керуються журналами вихідних завдань, щоб також додати цю функцію.	+	+	-	-

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Авто DevOps Авто DevOps приносить найкращі практики DevOps у ваш проект, автоматично налаштовуючи життєві цикли розробки програмного забезпечення за замовчуванням. Він автоматично виявляє, будує, тестує, розгортає та відстежує програми.	-	+	-	-
Захищені Runner Захищені Runners дозволяють захищати вашу конфіденційну інформацію, наприклад: облікові дані про розгортування, дозволяючи отримувати доступ до них лише робочим місцям, які працюють на захищених гілках.	-	+	-	+
Проста інтеграція існуючих кластерів Kubernetes Додайте існуючий кластер Kubernetes до свого проекту та легко перейдіть до нього зі своїх конвеєрів CI, щоб розмістити програми для перегляду та розгорнути свою програму.	-	+	-	-
Просте створення кластерів Kubernetes на GKE Створіть кластер Kubernetes на GKE безпосередньо зі свого проекту, просто підключіть свій обліковий запис Google і надайте деяку інформацію. Кластер може також використовуватися Auto DevOps для розгортування вашої програми.	-	+	-	-

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Підтримка декількох кластерів Kubernetes Легко розгортайте різні середовища, такі як Staging and Production, для різних кластерів Kubernetes. Це дозволяє забезпечити суворе розділення даних.	-	+	-	-
Просте розгортування програм для кластерів Kubernetes Встановіть Helm Tiller, Nginx Ingress, Cert-Manager, Prometheus, GitLab Runner, JupyterHub та Knative прямо у свій кластер із веб-інтерфейсу GitLab Web одним натисканням кнопки.	-	+	-	-
Канарські розгортування GitLab Enterprise Edition Premium може стежити за вашими розгортаннями Canary під час розгортання ваших програм за допомогою Kubernetes.	-	+	-	+
Мінімальна конфігурація CI GitLab CI вимагає меншої конфігурації для ваших трубопроводів, ніж інші подібні установки, такі як Jenkins.	-	+	-	-
Автоматичне повторне повторення роботи з помилкою CI Ви можете вказати ключове слово для повторного використання у вашому файлі .gitlab-ci.yml, щоб змусити GitLab CI повторно виконувати завдання певну кількість разів, перш ніж відзначати його як невдалий.	+	+	+	+

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Безпека трубопроводів Можливість запуску конвеєрів CI на захищених гілках перевіряється набором правил безпеки, який визначає, чи вам це дозволено чи ні. Він включає створення нових трубопроводів, повторну роботу та виконання ручних дій.	+	+	+	+
Включіть зовнішні файли у визначення конвеєра CI Ви можете включити зовнішні файли у файл визначення конвеєра, використовуючи їх як шаблони для повторного використання фрагментів для загальних завдань.				
Крок складання для журналів CI Згорнути вихід журналу завдань для кожної команди.	+	+	+	+
Перегляд журналів Kubernetes pod Моніторинг серверів, додатків, мереж та пристроїв безпеки через створені файли журналів для виявлення помилок та проблем для аналізу. GitLab дозволяє легко переглядати журнали запущених стручків у підключених кластерах Kubernetes. Показуючи журнали безпосередньо в GitLab, розробники можуть уникнути необхідності керувати консольними інструментами або переходити до іншого інтерфейсу.	-	+	-	-

Опис	Jenkins	Gitlab	Travis ci	CircleCi
Виконавець контейнерів Windows Завдяки цій функції ви можете використовувати контейнери Docker безпосередньо в Windows, майже так само, як якщо б вони були на хостах Linux. Це дає можливість більш вдосконаленим способом оркестрації конвеєрів та управління для користувачів платформ Microsoft.	+	+	+	+
Візуальні огляди Візуальний огляд дозволяє користувачам отримувати відгуки про запропоновану зміну в запиті на об'єднання, безпосередньо, з самого додатка огляду. Ця функція дозволяє дизайнерам, менеджерам продуктів та іншим зацікавленим сторонам, коментувати зміни в зовнішньому вигляді та користувальницькому досвіді змін.	-	+	-	-

1.2 Огляд технології віртуалізації і контейнерізації

Одна із проблем CI в тому, що інколи на машинах, що використовують розробники можуть відрізнятися версії бібліотек, з тими, що знаходяться на робочому сервері і для запобігання такого типу проблем, використовують ізольовані середовища - це можуть бути Docker або віртуальна машина

Віртуальна машина (VM) - це віртуальний комп'ютер, з усіма віртуальними пристроями і віртуальним жорстким диском, на який і встановлюється нова незалежна ОС (гостьова ОС), разом з віртуальними драйверами пристроїв, управлінням пам'яттю і іншими компонентами. Тобто, ми отримуємо абстракцію фізичного обладнання, що дозволяє запускати на одному комп'ютері безліч віртуальних комп'ютерів. Віртуальне обладнання відображається у властивостях системи, а встановлені програми взаємодіють з ним як з сьогоденням. При цьому сама віртуальна машина повністю ізольована від реального комп'ютера, хоча і може мати доступ до його диску і периферійних пристроїв.

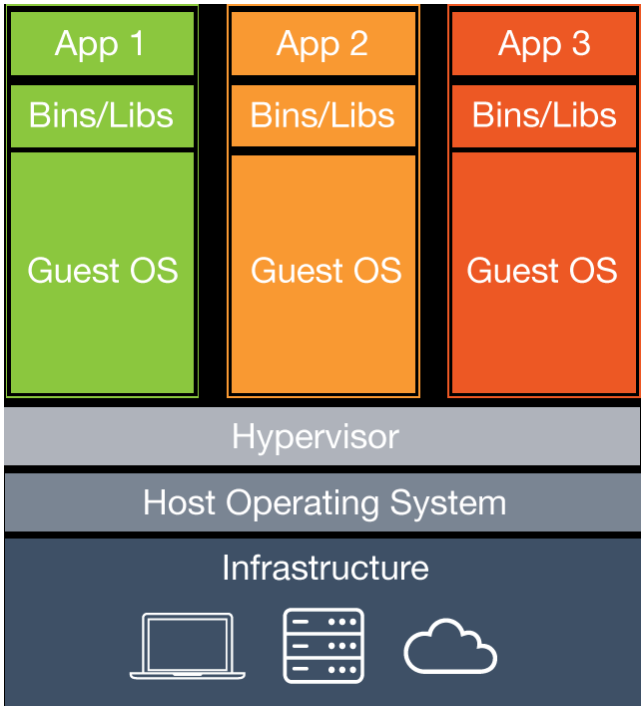

Рис. 1.1 Візуалізація роботи віртуальної машини

Встановлена VM може по-різному займати місце на диску комп'ютера:

- фіксоване місце на жорсткому диску, що дозволяє здійснювати більш швидкий доступ до віртуального жорсткого диска і дозволяє уникнути фрагментації файлу;
- динамічне виділення пам'яті. При встановленні додаткових додатків пам'ять буде динамічно виділятися під них, поки не досягне максимального обсягу, відведеного їй.

При використанні VM, з'являються додаткові витрати на емуляцію віртуального обладнання та запуск гостьовий ОС, підтримка і адміністрування необхідного оточення для роботи вашого додатка. Також, при розгортанні великої кількості віртуальних машин на сервері, обсяг займаного ними місця на жорсткому диску буде тільки рости, тому що для кожної VM потрібно місце, як мінімум, для гостьової ОС і драйверів для віртуальних пристроїв.

Docker - це ПЗ для створення додатків на основі контейнерів. Контейнери та віртуальні машини вирішують одну задачу, але роблять це по-різному. Контейнери займають менше місця, тому що використовують більшу кількість загальних ресурсів хост-системи ніж VM, тому що на відміну від VM, забезпечує віртуалізацію на рівні ОС, а не апаратного забезпечення. Такий підхід забезпечує менший обсяг займаного місця на жорсткому диску, швидке розгортування і більш просте масштабування.

Docker-контейнер дає більш ефективний механізм інкапсуляції додатків, забезпечуючи необхідні інтерфейси хост-системи. Дана можливість дозволяє контейнерів розділити ядро системи, де кожен з контейнерів працює, як окремий процес основної ОС, у якого є своя власна ВАП, таким чином дані, що належать різним областям пам'яті, не можуть бути змінені.

Docker найбільш поширена технологія використання контейнерів в роботі програми. Він став стандартом у цій галузі, працюючи на основі cgroups і namespaces, які забезпечує ядро Linux. Стандартна ОС для Docker - є Linux, тому запуск Docker-контейнерів на Windows, буде відбуватися всередині віртуальної машини з ОС Linux.

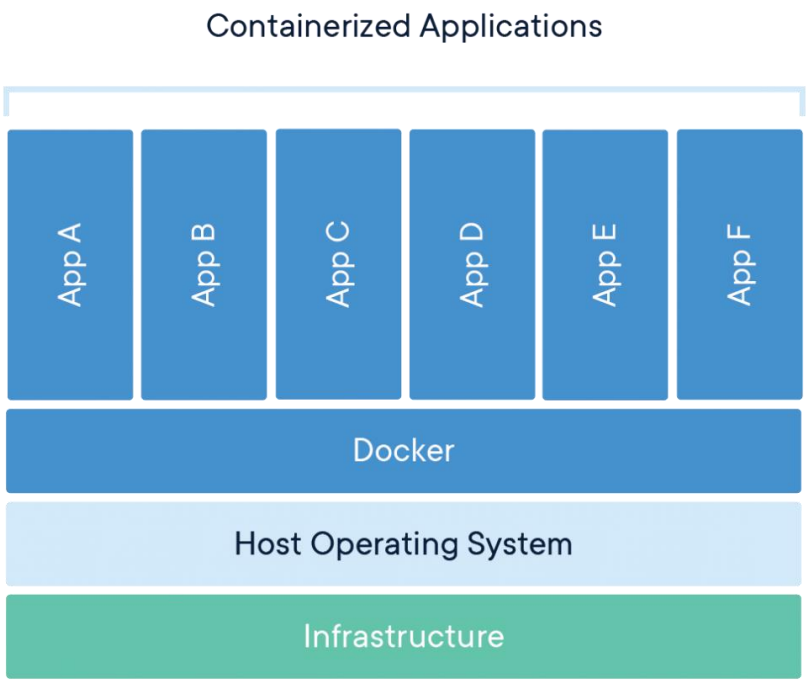

Рис 1.2 Візуалізація роботи контейнеру

Принцип роботи контейнеру

Зображення- основний елемент, з якого створюються контейнери. Образ створюється з Dockerfile, доданого в проект і являє собою набір файлових систем (шарів), накладаються один на одного і згруповаються разом, доступних тільки для читання: максимальне число шарів - 127.

В основі кожного образу знаходиться базовий образ, який вказується командою FROM - вхідна точка при формуванні образу Dockerfile. Кожен шар є readonly-шаром і представлений однією командою, що модифікує файлову систему, записаної в Dockerfile. Даний підхід дозволяє різним

файлам і директоріям, з різними файловими шарами прозоро накладатися, створюючи каскадно-об'єднану файлову систему. Шари містять метадані, що дозволяють зберігати супутню інформацію про кожній шар під час виконання і збірки. Кожен шар містить посилання на наступний шар, якщо шар не має посилання, це означає найвищий шар в образі.
Починаючи з версії Docker EE 17.06.02-ee5 і в Docker Engine - Community використовується Overlay або Overlay2, в більш ранніх версіях використовується AuFS (Advanced multi layered Union file system).

Контейнер - це абстракція на рівні додатку, що об'єднує код і залежності. Контейнери завжди створюються з образів, додаючи доступний для запису верхній шар і ініціалізує різні параметри. Так як контейнер має свій власний шар для запису і всі зміни зберігаються в цьому шарі, кілька контейнерів можуть спільно використовувати доступ до одного і того ж образу. Кожен контейнер можна налаштувати через файл в проекті docker-compose.yml, задаючи різні параметри, такі як ім'я контейнера, порти, ідентифікатори, залежності між іншими контейнерами. Якщо в налаштуваннях не ставити ім'я контейнера, то Docker кожен раз буде створювати новий контейнер, привласнюючи йому ім'я випадковим чином. Коли контейнер запускається з образу, Docker монтує файлову систему для читання і запису поверх будь-яких верств нижче. Саме тут будуть виконуватися всі процеси. При першому запуску контейнера, початковий шар читання-запису порожній. Коли відбуваються зміни, вони застосовуються до цієї верстви; наприклад, якщо є необхідність змінити файл, цей файл буде скопійований з нижнього шару тільки для read only в шар для читання і запису. Версія файлу, доступна тільки для читання, все ще буде існувати, але тепер вона прихована під копією.

Каскадно-об'єднана файлова система (ФС) реалізує механізм копіювання при записі (Copy-On-Write, COW). Робочою одиницею є шар, кожен шар слід розглядати як окрему повноцінну файлову систему з ієрархією директорій від самого кореня. Даний підхід використовує об'єднане монтування файлових систем, дозволяючи прозоро для користувача об'єднувати файли і каталоги різних файлових систем (званих гілками) в єдину пов'язану файлову систему. Вміст каталогів з однаковими шляхами буде відображатися разом в одному об'єднаному каталозі (в єдиному просторі назв) отриманої файлової системи.

Об'єднання шарів відбувається за такими принципами:

- один з шарів стає шаром верхнього рівня, другий і наступні - шарами нижнього рівня;
- користувачеві, об'єкти шарів доступні «зверху вниз», тобто якщо запитаний об'єкт є в «верхньому» шарі, повертається він, незалежно від наявності об'єкта з таким ім'ям в «нижньому» шарі; інакше повертається об'єкт «нижнього» шару; якщо запитаного об'єкта немає ні там, ні там, повертається помилка «Немає такого файлу або каталогу»;
- робочим шаром є «верхній», тобто всі дії користувача, зі зміни даних відображаються тільки на шарі верхнього рівня, не впливаючи на вміст шарів нижніх рівнів.

Висновки до розділу 1

Системи атоматизації

Якщо підсумувати, то Gitlab має свій клауд, проте скоріш за все, команда розробників буде використовувати сторонні сервери, через обмежений безкоштовний час роботи сервера та вбудованій Git, що надає велику перевагу в зручності використання, але Jenkins має величезний вибір плагінів хоча ці плагіни й написані сторонніми розробниками і не гарантовано працюють - є можливість завжди написати свої. Тому, можна сказати, що обидва додатки мають свої сильні й слабкі сторони і кожен підходить для різних видів розробки додатків. Travis Ci I CircleCI мають закритий код і зв’язок з розробниками дуже обмежений, тому їх не варто використовувати в нових проектах, лише в тому випадку, якщо на проекті не буде людини, що зможе займатися CI.

Docker і VM

При необхідності віртуалізації системи з гарантовано виділеними ресурсами і віртуальним апаратним забезпечення, варто вибрати VM. Що дає використання VM:

- можливість установки на одному комп'ютері декількох різних ОС;
- розподіл системних ресурсів між віртуальними машинами;
- відсутність необхідності перезавантаження для перемикання між операційними системами;
- можливість зробити «знімок» поточного стану системи і вмісту дисків, для повернення системи в початковий стан;
- ізоляція несправностей і порушень системи безпеки на апаратному рівні;
- можливість моделювання обчислювальної мережі на одному комп'ютері

Якщо ціллю є ізолювати працюючі додатки, як окремі процеси, то підійде Docker. Що дає використання Docker:

- забезпечує віртуалізацію на рівні ОС;
- контейнери поділяють ядро системи, працюючи як окремий процес основний ОС;
- споживання системних ресурсів, таких як витрата пам'яті і навантаження на CPU, можуть обмежуватися окремо для кожного контейнера з використанням cgroups;
- ФС для контейнерів створюється з використанням механізму COW, що дозволяє прискорити розгортування додатки, знижує витрату пам'яті і економить місце на диску;
- змінена файлова система одного контейнера, може використовуватися в якості основи для формування нових базових образів інших контейнерів.

РОЗДІЛ 2

КЛАДУДИ і Blue/Green Deployment

2.1. Існуючі клауди та їх порівняння

Багато експертів рекомендують підприємствам, оцінювати свої суспільні потреби у хмарі, залежно від конкретного випадку, узгоджувати конкретні програми та навантаження з постачальником, який найкраще відповідає їх потребам. Кожен з провідних постачальників, має особливі сильні та слабкі сторони, які роблять їх хорошим вибором для певних проектів.

Переваги і недоліки AWS

На цей момент, AWS - є найпопулярнішим постачальником хмарних технологій. Частково причиною його популярності - є масштабність його діяльності. AWS має величезний і постійно зростаючий набір доступних послуг, а також найбільшу мережу світових дата-центрів. У звіті Gartner підсумовується: "AWS - найбільш зрілий, готовий до роботи на підприємстві провайдер, що володіє широкими можливостями управління великою кількістю користувачів і ресурсів".

Найбільша слабкість Amazon пов'язана з витратами. У той час як AWS регулярно знижує ціни, багатьом підприємствам важко зрозуміти структуру витрат компанії та ефективно керувати цими витратами при великій кількості робочих навантажень на сервіс.

В цілому, однак, ці недоліки більш ніж переважують сильні сторони Amazon, і організації всіх розмірів продовжують використовувати AWS для виконання найрізноманітніших завдань.

Переваги і недоліки Microsoft Azure

Компанія Майкрософт прийшла на ринок хмарних технологій доволі пізно, але знайшла популярність взявши своє місцеве програмне

забезпечення - Windows Server, Office, SQL Server, Sharepoint, Dynamics Active Directory, .Net та інші і перепрофілювавши його на хмарні версії.

Більшість успіхів Azure зв’язано з тим, що багато інших підприємств встановлюють Windows та інші програми Microsoft. Через те, що Azure тісно інтегрується з цими та іншими додатками, підприємства, що використовують велике число програмних послуг Microsoft, часто використовують і Azure. Це підвищує лояльність існуючих клієнтів Microsoft. У той же час, якщо ви вже представляєте фактично корпоративного клієнта Microsoft, ви отримаєте значні знижки за контрактами на обслуговування.

Переваги і недоліки GCP

Google має сильну пропозицію в контейнерах, так як компанія Google розробляє стандарт Kubernetes, який в даний час пропонує AWS та Azure. GCP спеціалізується на пропозиціях для великих обчислень, таких як "Big Data", аналітики та машинного навчання. Окрім того, пропонується балансування навантаження.

Недоліком є те, що Google займає третю позицію на ринку, можливо, тому що він не пропонує велику різноманітність послуг та функцій, як AWS та Azure. Крім того, Google не пропонує так багато глобальних центрів обробки даних, як у AWS або Azure, але він швидко розширюється.

Табл 2.1 порівняння хмарних постачальників

Постачальник	Сильні сторони	Слабкі місця
AWS	• Домінуюча позиція на ринку • Широкі, зрілі пропозиції • Підтримка великих організацій • Поглиблене навчання	• Важкий у використанні • Управління витратами
Microsoft Azure	• Другий за величиною постачальник • Інтеграція з інструментами та програмним забезпеченням Microsoft • Широкий набір функцій • Гібридна хмара	• Проблеми з документацією • Недостаточний інструментарій управління
Google	• Призначений для хмарного бізнесу • Прихильність до open source та портативності • Глибокі знижки та гнучкі договори	• Пізній учасник ринку IaaS • Менше функцій та послуг

Обчислення

AWS Compute:

- **Elastic Compute Cloud:** Флагманський обчислювальний сервіс Amazon - це Elastic Compute Cloud, або EC2. Amazon описує EC2 як "веб-сервіс, що забезпечує безпечні обчислення з можливістю зміни розміру в хмарі". EC2 пропонує широкий вибір варіантів, включаючи величезний асортимент примірників, підтримку як Windows, так і Linux, екземпляри GPU, високопродуктивні обчислення, автоматичне масштабування і багато іншого. AWS також пропонує безкоштовний рівень для EC2, який включає 750 годин на місяць до дванадцяти місяців.

- **Container services:** У обчислювальної категорії все більшої популярності набувають різні контейнерні сервіси Amazon, які підтримують Docker, Kubernetes і власний сервіс Fargate, що автоматизує управління серверами і кластерами при використанні контейнерів. Він також пропонує опцію віртуальної приватної хмари, відому як Lightsail, Batch для пакетних обчислень, Elastic Beanstalk для запуску і масштабування веб-додатків, а також деякі інші сервіси.

Microsoft Compute:

- **Виртуальні машини:** Основна обчислювальна служба Microsoft відома просто, як Віртуальні машини. Вона може похвалитися підтримкою Linux, Windows Server, SQL Server, Oracle, IBM і SAP, а також підвищеною безпекою, гібридними хмарними можливостями і інтегрованою підтримкою програмного забезпечення Microsoft. Як і AWS, він має надзвичайно великий каталог доступних екземплярів, включаючи GPU і високопродуктивні обчислення, а також екземпляри, оптимізовані для штучного інтелекту і машинного навчання. Він також

має безкоштовний рівень з 750 годинами на місяць віртуальних машин Windows або Linux B1S протягом року.

- **Додадткові послуги:** Версія Azure для автоматичного масштабування відома як Virtual Machine Scale Sets. У ній є два контейнерних сервісу: Azure Container Service базується на Kubernetes, а Контейнерні послуги використовують Dokerhub і Регістр контейнерів Azure для управління. Вона має пакетну службу, а хмарні служби для масштабованих веб-додатків схожі на AWS Elastic Beanstalk. Вона також має унікальну пропозицію під назвою Service Fabric, яке спеціально розроблене для додатків з архітектурою мікро-сервісів.

Google Compute:

- **Обчислювальна машина:** Для порівняння, каталог обчислювальних послуг Google дещо коротший, ніж у його конкурентів. Його основний сервіс називається - Compute Engine, який може похвалитися різними типами машин, посекундним виставленням рахунків, підтримкою Linux і Windows, автоматичними знижками і вуглецево-нейтральної інфраструктурою, яка використовує половину енергії типових центрів обробки даних. Він пропонує безкоштовний рівень, що включає один екземпляр f1-мікро в місяць на термін до 12 місяців.

- **Kubernetes:** Google також пропонує Kubernetes Engine для організацій, зацікавлених в розгортанні контейнерів. Як і всі провідні постачальники "хмарних" обчислень, вона налаштована на надання контейнерів та мікроуслуг. І варто зазначити, що компанія Google брала активну участь в проекті Kubernetes, надавши їй додаткову експертизу в цій області.

Табл 2.2 порівняння хмарних обчислень

Постачальник	Комп'ютерні послуги
AWS	EC2 • Elastic Container Service • Elastic Container Service for Kubernetes • Elastic Container Registry • Lightsail • Batch • Elastic Beanstalk • Fargate • Auto Scaling • Elastic Load Balancing
Microsoft Azure	Virtual Machines • Virtual Machine Scale Sets • Azure Container Service (AKS) • Container Instances • Batch • Service Fabric
Google Cloud	Compute Engine • Kubernetes • Functions • Container Security • Graphics Processing Unit (GPU) • App Engine

Зберігання данних

AWS Storage:

- **SSS to EFS:** AWS пропонує довгий список сервісів зберігання, що включає Simple Storage Service (S3) для зберігання об'єктів, Elastic Block Storage (EBS) для постійного блочного зберігання, використання з EC2 і Elastic File System (EFS) та зберігання файлів. Серед його більш інноваційних продуктів в області зберігання даних, можна назвати шлюз Storage Gateway, що забезпечує гібридну середу зберігання, і Snowball, який є фізичною апаратним пристроєм, що організації можуть використовувати для передачі петабайт даних в ситуаціях, коли передача даних через Інтернет недоцільна.

- **Database and archiving:** На стороні бази даних Amazon має SQL-сумісну базу даних Aurora, Relational Database Service (RDS), базу даних DynamoDB NoSQL, сховище даних в пам'яті ElastiCache, сховище даних Redshift, базу даних графів Neptune і службу міграції бази даних. Amazon пропонує Glacier, який призначений для довгострокового архівного зберігання даних на дуже низьких швидкостях. Крім того, його шлюз Storage Gateway можна використовувати для простого налаштування процесів, резервного копіювання та архівування.

Azure Storage:

- **Storage Services:** Основні послуги Microsoft Azure зі зберігання даних включають: Blob Storage для REST-об'єктного зберігання неструктурованих даних, Queue Storage для робочих навантажень великого обсягу, File Storage та Disk Storage. Вона також має Data Lake Store, який корисний для великих додатків даних.

- **Обширная база данных:** Варіанти бази даних Azure особливо великі. У ній є три варіанти на основі SQL: SQL Database, Database for MySQL і Database for PostgreSQL. У ній також є служба Data Warehouse, а також Cosmos DB і Table Storage for NoSQL. Redis Cache - це служба внутрішньої пам'яті, а Server Stretch Database - це гібридна служба зберігання, розроблена спеціально для організацій, які використовують Microsoft SQL Server в своїх власних центрах даних. На відміну від AWS, Microsoft дійсно пропонує актуальну службу Резервного копіювання, а також службу відновлення сайтів та архівного зберігання.

Google Storage:

- **Єдине сховище і багато іншого:** Як і у випадку з обчисленнями, в GCP є менша кількість сервісів зберігання. Хмарне сховище - це його уніфікований сервіс зберігання об'єктів, а також має опцію "Постійний диск". Воно пропонує Трансферне Пристрій, аналогічне AWS Snowball, а також послуги онлайн-трансферу.

- **SQL и NoSQL:** Коли мова йде про бази даних, GCP має SQL на основі Cloud SQL і реляційну базу даних під назвою Cloud Spanner, яка призначена для критично важливих робочих навантажень. У нього також є два варіанти NoSQL: Cloud Bigtable і Cloud Datastore. У нього немає сервісів резервного копіювання та архівування.

Табл 2.3 порівняння хмарних сховищ

Постачальник	Послуги по зберіганню	Послуги баз данних	Backup Services
AWS	Simple Storage Service (S3) • Elastic Block Storage (EBS) • Elastic File System (EFS) • Storage Gateway • Snowball • Snowball Edge	Aurora • RDS • DynamoDB • ElastiCache • Redshift • Neptune	Glacier
Azure	• Blob Storage • Queue Storage • File Storage • Disk Storage • Data Lake Store	• Database for MySQL • Database for PostgreSQL • Data Warehouse • Server Stretch Database • Cosmos DB • Table Storage • Redis Cache • Data Factory	• Archive Storage • Backup • Site Recovery
GCP	•Cloud Storage • Persistent Disk • Transfer Appliance • Transfer Service	•Cloud SQL • Cloud Bigtable • Cloud Spanner • Cloud Datastore	None

Ключові хмарні інструменти

Ключові хмарні інструменти AWS:

- **Pagemaker to Serverless:** Як і в інших областях, AWS має найдовші списки послуг в кожній з цих областей. Серед основних моментів - сервіс SageMaker для навчання і розгортування моделей машинного навчання, розмовний інтерфейс Lex, який також підтримує сервіси Alexa, служба повідомлень Greengrass IoT і служба бессерверной обчислень Lambda.
- **AI и ML:** Серед численних сервісів, орієнтованих на штучній інтелект, AWS пропонує DeepLens для розробки і впровадження алгоритмів машинного навчання, які можна використовувати з такими речами, як оптичне розпізнавання символів, розпізнавання зображень і об'єктів. AWS оголосила про випуск Gluon - бібліотеки машинного навчання призначеної для того, щоб розробники та не розробники могли легко будувати і швидко навчати нейронні мережі, не розбираючись в програмуванні під штучній інтелект.

Azure Key Tools:

- **Cognitive Services:** Microsoft також інвестувала значні кошти в штучний інтелект, і вона пропонує службу машинного навчання, і службу роботів на Azure. Вона також пропонує когнітивні послуги, які включають Bing Web Search API, Text Analytics API, Face API, Computer Vision API і Custom Vision Service. Для IoT, він має кілька послуг управління та аналізу, і його безсерверний обчислювальний сервіс відомий як Functions.
- **Підтримка програмного забезпечення MSFT:** Не дивно, що більшість з кращих інструментів компанії Azure орієнтовані на підтримку локального програмного забезпечення Microsoft. Azure Backup - це служба, яка пов'язує Windows Server Backup в Windows

Server 2012 R2 і Windows Server 2016. Visual Studio Team Services розміщує проекти Visual Studio на Azure.

Google Key Tools:

- **Увага до ШІ:** Для Хмарної платформи Google ШІ і машинне навчання є приорітетними напрямками. Google є лідером в розробці ШІ завдяки TensorFlow, програмної бібліотеки з відкритим вихідним кодом з метою створення додатків для машинного навчання. Бібліотека TensoreFlow популярна і користується гарною репутацією. Свідченням її популярності є те, що AWS недавно додала підтримку TensorFlow.
- **IoT - Serverless:** Google Cloud має сильні пропозиції в API для мови, перекладу і багато чого іншого. Крім того, він пропонує IoT і безсерверной послуги, але і ті і інші все ще перебувають в бета-тестуванні

Табл 2.4 порівняння ключових хмарних технологій

Продавец	AI/ML	Internet of things	Serverless
AWS	•SageMaker •Comprehend • Lex • Polly •Rekognition •Machine Learning • Translate •Transcribe •DeepLens • Deep Learning AMIs • Apache MXNet on AWS • TensorFlow on AWS	• oT Core •FreeRTOS •Greengrass • IoT 1-Click • IoT Analytics • IoT Button • IoT Device Defender • IoT Device Management	• Lambda • Serverless Application Repository
Azure	• Machine Learning • Azure Bot Service • Cognitive Services	• IoT Hub • IoT Edge • Stream Analytics • Time Series Insights	·Functions
GCP	• Cloud Machine Learning Engine • Dialogflow Enterprise Edition • Cloud Natural Language • Cloud Speech API • Cloud Translation API • Cloud Video Intelligence • Cloud Job Discovery (Private Beta)	·Cloud IoT Core (Beta)	·Cloud Functions (Beta)

Ціни

При порівнянні трьох лідерів "хмар", ціноутворення іноді є найхитріша з усіх областей. Проте, можна зробити деякі узагальнення.

- **Ціноутворення AWS:** Ціноутворення в AWS особливо складне. Незважаючи на те, що він пропонує калькулятор вартості, велика кількість задіяних змінних, ускладнює отримання точних оцінок. Настійно рекомендується, використання сторонніх інструментів управління витратами".
- **Ціноутворення Azure:** Microsoft Azure не робить ці речі простіше. Через складні варіанти ліцензування програмного забезпечення Microsoft і використання ситуаційних знижок, структура ціноутворення Microsoft може бути складна для розуміння без сторонньої допомоги або значного досвіду.
- **Ціноутворення Google:** Google, навпаки, використовує своє ціноутворення як точку диференціації. Вона прагне запропонувати "зручні для клієнтів" ціни, які перевершують ціни інших провайдерів. Google використовує глибокі знижки та виключно гнучкі контракти, щоб спробувати виграти проекти у клієнтів, які в даний час витрачають значні суми грошей з хмарними конкурентами.

2.2. Blue/Green Deployment

Однією з проблем автоматизації розгортання є саме скорочення, переклад програмного забезпечення з фінальної стадії тестування на живе виробництво. Зазвичай потрібно зробити це швидко, щоб мінімізувати час простою. Синьо-зелений підхід до розгортання робить це, забезпечуючи дві однакові виробничі середовища. У будь-який момент, скажімо, блакитний, наприклад, живий. Коли розробник готуєте нову версію свого програмного

забезпечення, він проходить фінальну стадію тестування в зеленому середовищі. Коли програмне забезпечення працює в зеленому середовищі, розробник переключає маршрутизатор, щоб всі вхідні запити перейшли в зелену середу – блакитний тепер не діє.

Блакитно-зелене розгортання також надає швидкий спосіб відкату - якщо щось піде не так, переведіть маршрутизатор назад в блакитну середу. До сих пір існує проблема обробки пропущених транзакцій в той час, коли зелене середовище працювало, але в залежності від дизайну можна передавати транзакції в обидва середовища таким чином, щоб зберегти блакитну середу в якості резервної копії, коли зелена серед активних. Або можна перевести додаток в режим «тільки читання» перед тим, як його відключити, а потім переключити в режим «читання і запис». Цього може бути достатньо, щоб позбутися від багатьох невирішених питань.
Ці два середовища повінні буті різні, але максимально ідентичні. У деяких сітуаціях, це можуть бути різні апаратні части або різні віртуальні машини, что працюють на одному й тому ж (або різному) обладнанні.
Після того, як було запущено «зелену» середу і команда задоволена її стабільністю, можна використовувати «блакитну», в якості проміжного середовища, з метою заключного етапу тестування для наступного розгортання. Перехід з зеленого на блакитний такий самий, як з блакитного на зелений. Таким чином, зелене та блакитне середовище регулярно переключаються між поточною, попередньою версією і підготовкою наступної версії.

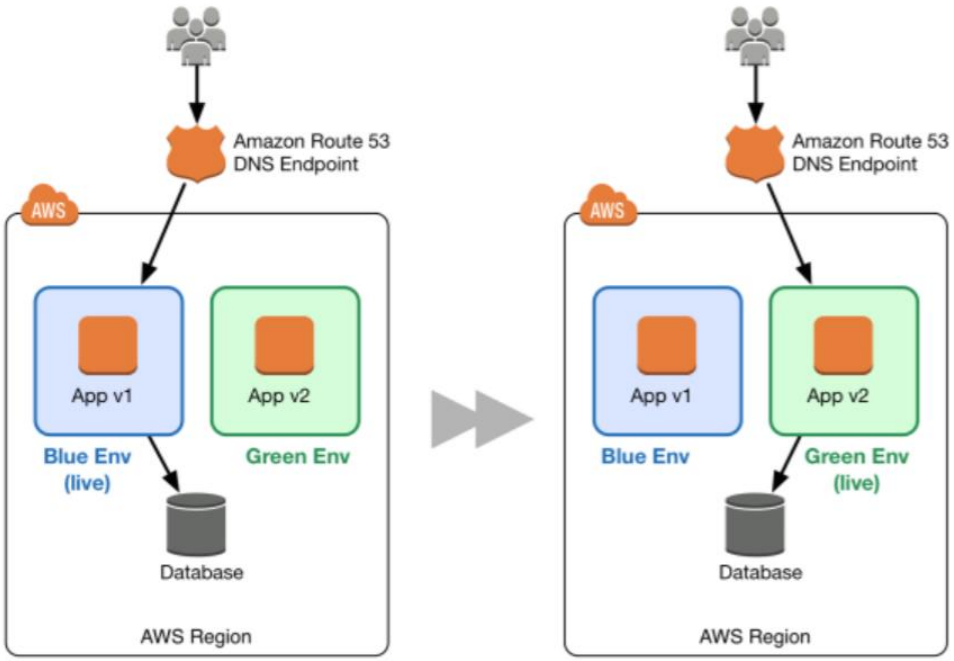

Рис 2.1 Візуалізація роботи Blue/Green

Перевага цього підходу в тому, що це, той же базовий механізм, який необхідний для роботи в режимі гарячого резервування. Отже, це дозволяє тестувати процедуру аварійного відновлення на кожному випуску. Основна ідея полягає в тому, щоб мати два легко перемикаючі середовища для перемикання, існує безліч способів варіювати деталі. Один проект зробив перемикання шляхом підстрибування веб-сервера, а не роботи на маршрутизаторі. Інший варіант - використовувати ту ж базу даних, роблячи блакитно-зелені перемикачі для веб-шарів і прошарків домену.

Бази даних часто можуть мати проблемою з цим методом, особливо, коли вам потрібно змінити схему для підтримки нової версії програмного забезпечення. Хитрість полягає в тому, щоб відокремити розгортання змін схеми від оновлень програми. Тому спочатку, застосовуєте рефакторинг бази даних, щоб змінити схему для підтримки нової і старої версії додатка, розгорніть її, перевірте, що все працює нормально, щоб була точка відкату, а потім розгорніть нову версію програми.

Висновки до розділу 2

Кращий постачальник публічного хмари, буде залежати від потреб замовника і робочих навантажень. Насправді, найкращий постачальник для деяких проектів може бути не кращим для інших проектів. Багато експертів вважають, що більшість підприємств будуть вкладати значні кошти в мульти-хмару. Дійсно, реалізація стратегії мульти-хмари, можливо, допоможе знизити залежність від постачальника або співвіднести робочі навантаження з найкращим доступним сервісом.

- **Вибір AWS:** Обравши AWS складно помилитися через багату колекцію інструментів, сервісів і величезних масштабів. Єдина причина не вибирати Amazon - це якщо замовник хоче мати більш особисті стосунки з постачальником хмари. За своїми розмірами Amazon складно мати тісні відносини з кожним клієнтом, але є реселлери і консультанти, які можуть запропонувати такий уважний підхід.
- **Вибір Azure:** Найбільша привабливість Microsoft, звичайно ж, для магазинів Microsoft. Весь існуючий код .Net буде працювати на Azure, серверна середовище буде підключатися до Azure, і існує можливість легко переносити локальні програми. Futhermore, глибока орієнтація Azure на гібридне "хмара" допоможе поєднати успадковану середу центру обробки даних, з швидко масштабованою "хмарою" Microsoft.
- **Вибір Google:** Google швидко розвивається, але робота триває. Природньо, що у пошукового гіганта немає досвіду роботи з бізнесом. Але вона повністю віддана своїй справі і вклала мільярди у свої "хмарні" зусилля. І вона співпрацює з Cisco, яка дійсно знає підприємство. Люди, які повинні дивитися на Google зараз, - це ті, хто дивився рік тому і їм не сподобалося те, що вони побачили. Вони можуть бути здивовані. Google побудував свою хмару зі своїми сильними сторонами, якими є масштаб і машинне навчання.

Blue/Green Deployment допомагає вирішити величезну кількість проблем, пов’язану з розгортуванням нової версії, проте цей підхід має деякі мінуси: по-перше - база данних повинна бути на сторонньому сервері, по-друге - можливо великий цінник, якщо проект зробленій не з сервісною архітектурою. Тому треба розуміти, що цей тип розгортування, не заміщує собою інші, але має свої сильні сторони і його варто використовувати у проектах.

РОЗДІЛ 3

МОДЕЛЮВАННЯ ЗАПРОПОНОВАНОГО АЛГОРИТМУ

Для моделювання запропонованого алгоритму було обрано зв’язку GitLab + docker + GCP + Blue/Green. Цей вибір слід обґрунтувати наступними чинниками:

- GitLab має безкоштовний варіант використання в клауді GitLab, таким чином є можливість не витрачати кошти на використання додаткового серверу. Також GitLab і Jenkins мають дуже схожу архітектуру будування пайплайнів і це спрощує міграцію між ними, якщо з часом це стане необхідним.
- Docker дуже зручний в використанні, а також потребує невелику кількість ресурсів, що робить його дуже економічним для роботи в клауді. До того ж, всі клауди мають можливість використовувати безсерверну архітектуру на базі контейнерів, що ще більше зменшує витрати.
- GCP надає користувачам безкоштовну версію, як і всі інші клауди, але також надає кредит в 300$ на реалізацію ресурсів понад можливості безкоштовної версії, що робить його максимально економічно вигідним для реалізації заданого алгоритму.

При розробці програми, використовується операційна система Linux з дистрибутивом Ubuntu, для можливості тестувати Docker контейнери і зручного користування Git

При роботі, бажано використовувати наступні технічні характеристики комп’ютерної системи:

- процесор з 2 ядрами тактовою частотою не менше ніж 2 ГГц;
- оперативна пам’ять об’ємом не менше ніж 4 Гб;

Інші характеристики не мають вагомого впливу для коректної роботи.

3.1. Опис логіки програми

Продукт має надавати можливості для роботи з наступними параметрами:

- Мінімальна ціна;
- Мінімальний час від збірки до розгортання;
- Зручність в використанні

Отже, розглянемо структуру процесу автоматизації:

- Створення аккаунту в GitLab
- Створення нового проекту в GitLab
- Створення аккаунту а потім і нового проекту GCP
- Створення нового репозіторію в Google container registry
- Додавання до проекту Dockerfile, що описує процесс створення контейнеру
- Додавання до проекту файлу .gitlab-ci.yml, що описує процес створення з Dockerfile контейнера з готовою програмою в середині, запускає автоматичні тести та відправляє проект в GCR
- Push проекту в репозіторій GitLab, після чого автоматично запуститься виконнаня .gitlab-ci.yml
- Після завершення виконання пайплайну, контейнер, якщо він успішно пройшов всі тести, з'явиться в GCR
- Далі потрібно перейти в сервіс Cloud run та створити новий сервіс, що й буде контейнером
- Для реалізації Blue/Green потрібно: після оновлення запустити новий контейнер, але не пускати на нього трафік, та через деякий час, поступово, змінювати трафік зі старої версії на нову, через це не всі користувачі одразу побачать зміни на сервері, але команда розробників зможе контролювати весь процес деплою , і в разі невдачі, швидко зупинити розгортання, і лише невелика частина користувачів побачать помилку. Після успішного розгортання старий контейнер можна зупинити.

3.2. Приклад роботи

GitLab

https://gitlab.com

Створення аккаунту в GitLab

Перейти https://gitlab.com/users/sign_in

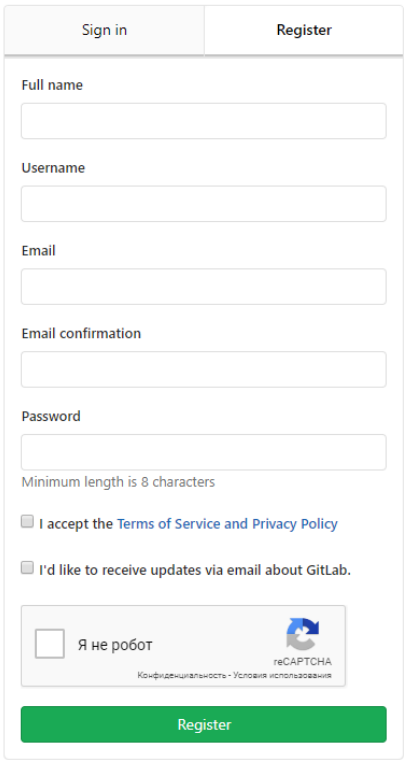

Рис 3.1 Поля регістрації в Gitlab

Записати данні в поля вводу (Рис 3.1) і натиснути Register. Також є можливість автоматично зареєструватись за допомогою (Рис 3.2)

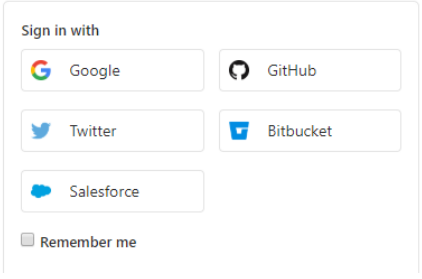

Рис 3.2 Перелік доступних систем авторегістрації

Після чого, GitLab автоматично надасть вам можливість використовувати функціонал безкоштовного аккаунту.

Створення нового репозіторію в GitLab

Авторизований користувач при переході на https://gitlab.com побачить наступне (Рис 3.3)

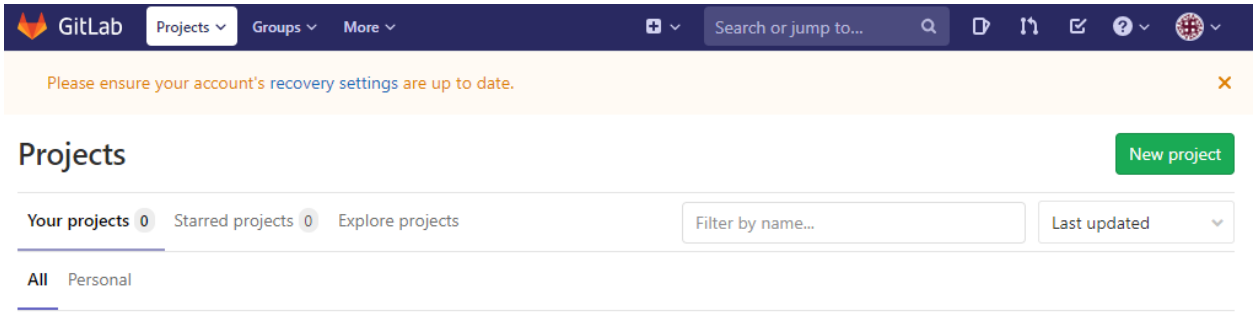

Рис 3.3 Вікно активних проектів

Для створення нового проекту, потрібно натиснути кнопу **New project**

Після натискання, користувач побачить наступене (Рис 3.4).

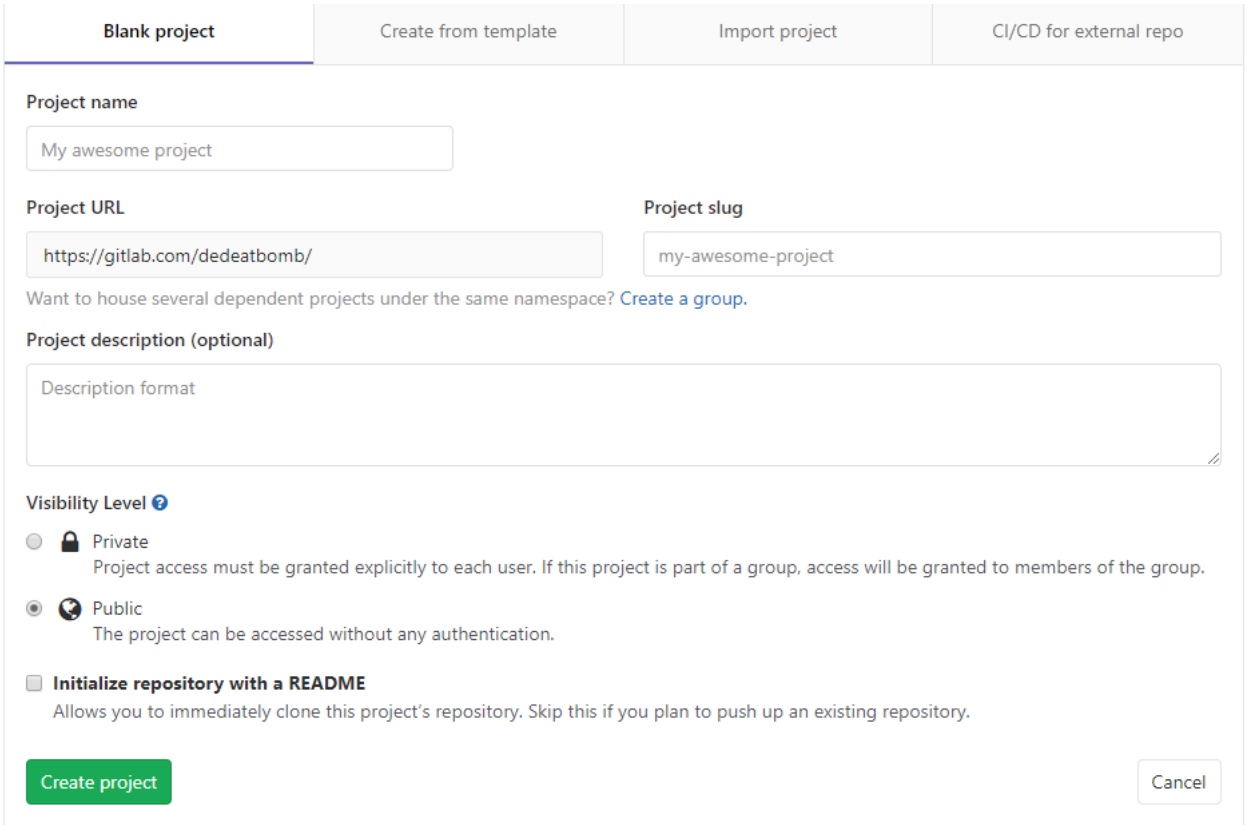

Рис 3.4 Вікно створення нового пректу

- Project name – поле вводу для назви проекту
- Project URL – адрес проекту
- Project slug – поле вводу для програмної назви проекту (при написанні назви проекту, автоматично заповнюється)
- Project description (optional) – видимість проекту для всіх користувачів (безкоштовний варіант включає лише публічну версію)

Після введення потрібних данних, користувачу потрібно натиснути Create Project

Наступним, що побачить користувач буде (Рис 3.5).

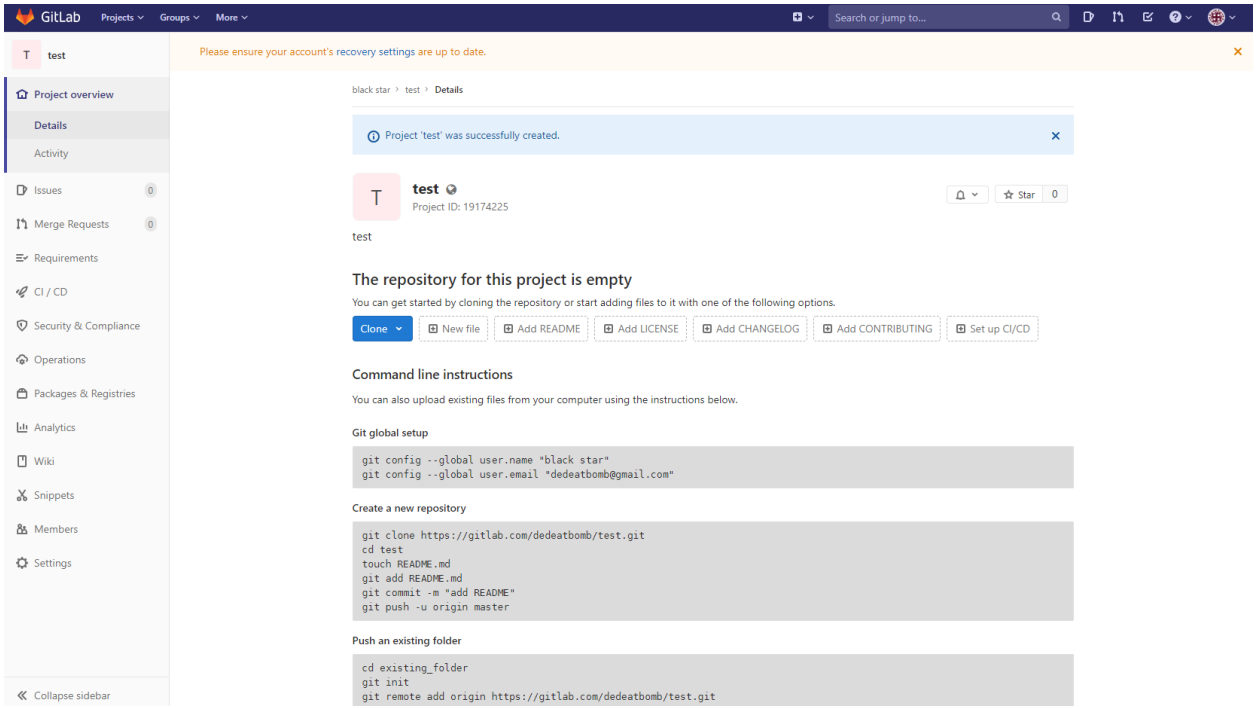

Рис 3.5 Вікно нового пректу

Зліва, можна побачити велику кількість розділів і записів

Рис 3.6 інструкція по додаванню проекту в Gitlab

Після виконання всіх дій зазначених в інструкції (Рис 3.6), користувач побачить

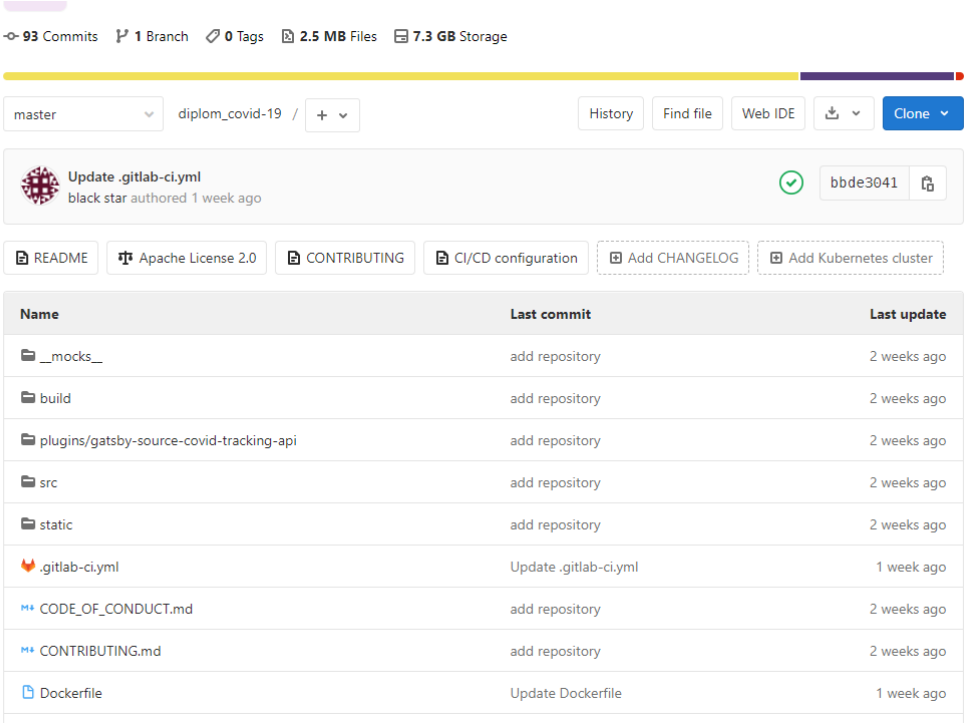

Рис 3.7 Доданий проект в Gitlab

Це репозиторій проекту зі всіма файлами (Рис 3.7).

Після додавання проекту в Gitlab, автоматично запуститься виконання файлу .gitlab-ci.yml

Зміст .gitlab-ci.yml:

```
image: docker:latest

stages:
  - build
  - test
  - deploy

variables:
  DOCKER_TLS_CERTDIR: ""

services:
  - docker:dind

.teamplate:
  script:
    - docker run -i --name $DIPLOM_CONTAINER $DIPLOM_IMAGE sh -c
"$SCRIPT"
    - docker commit $DIPLOM_CONTAINER $DIPLOM_ARCHIVE
    - if ! [ -d ./image ]; then mkdir image; fi
    - docker save $DIPLOM_ARCHIVE > image/$DIPLOM_ARCHIVE.tar
  artifacts:
    paths:
      - image/$DIPLOM_ARCHIVE.tar
    expire_in: 1 hrs

build:
  stage: build
  extends: .teamplate
  variables:
    DIPLOM_IMAGE: "my-diplom-image"
    DIPLOM_CONTAINER: "my-diplom-container"
    SCRIPT: 'npm install -g gatsby-cli && npm install && npm run setup:api-data
&& npm run setup'
    DIPLOM_ARCHIVE: "my-diplom-tar"
  before_script:
    - docker build -t $DIPLOM_IMAGE .

test:
```

```
  stage: test
  extends: .teamplate
  variables:
    DIPLOM_IMAGE: "my-diplom-tar"
    DIPLOM_CONTAINER: "my-diplom-container-test"
    SCRIPT: 'npm run test'
    DIPLOM_ARCHIVE: "my-diplom-test-tar"
  before_script:
    - docker load -i image/$DIPLOM_IMAGE.tar

deploy:
  stage: deploy
  variables:
    DIPLOM_ARCHIVE: "image/my-diplom-test-tar.tar"
    DIPLOM_IMAGE: "my-diplom-test-tar"
  before_script:
    - apk add --update python
    - apk add --update curl
    - apk add --update which
    - apk add --update bash
    - curl -sSL https://sdk.cloud.google.com | bash
    - PATH=$PATH:/root/google-cloud-sdk/bin
    - gcloud auth activate-service-account $IAM --key-file=$KEY
  script:
    - docker load -i $DIPLOM_ARCHIVE
    - docker tag $DIPLOM_IMAGE $ECR
    - gcloud docker -- push $ECR
```

Зміст Dockerfile:

```
FROM alpine:latest

COPY  ./ /diplom/

WORKDIR /diplom

RUN apk add npm && apk add curl
```

Процес виконання пайплайну можна побачити у вкладці CI/CD Piplines (Рис 3.8)

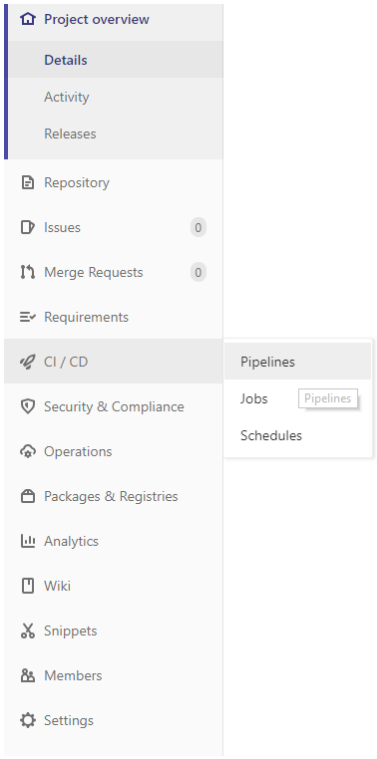

Рис 3.8 Ліва бокова панель в Gitlab

Перейшовши в цю вкладку користувач побачить (Рис 3.9)

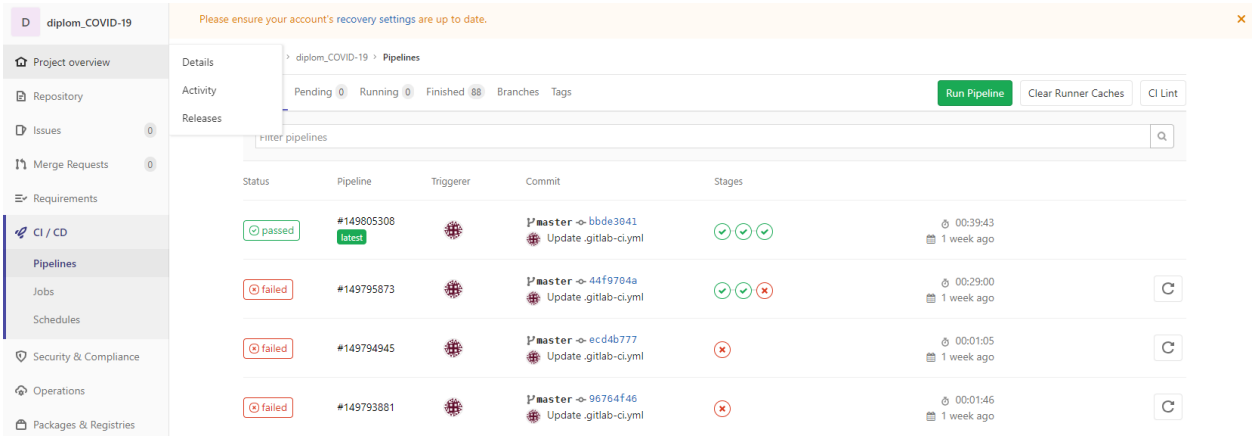

Рис 3.9 Перелік пайплайнів та їх статус в Gitlab

Статус пайплайну passed означає, що пайплайн успішно завершився

GCP

https://console.cloud.google.com

Для реєстрації в GCP потрібно створити акаунт в гугл, для цього потрібно перейти на https://accounts.google.com

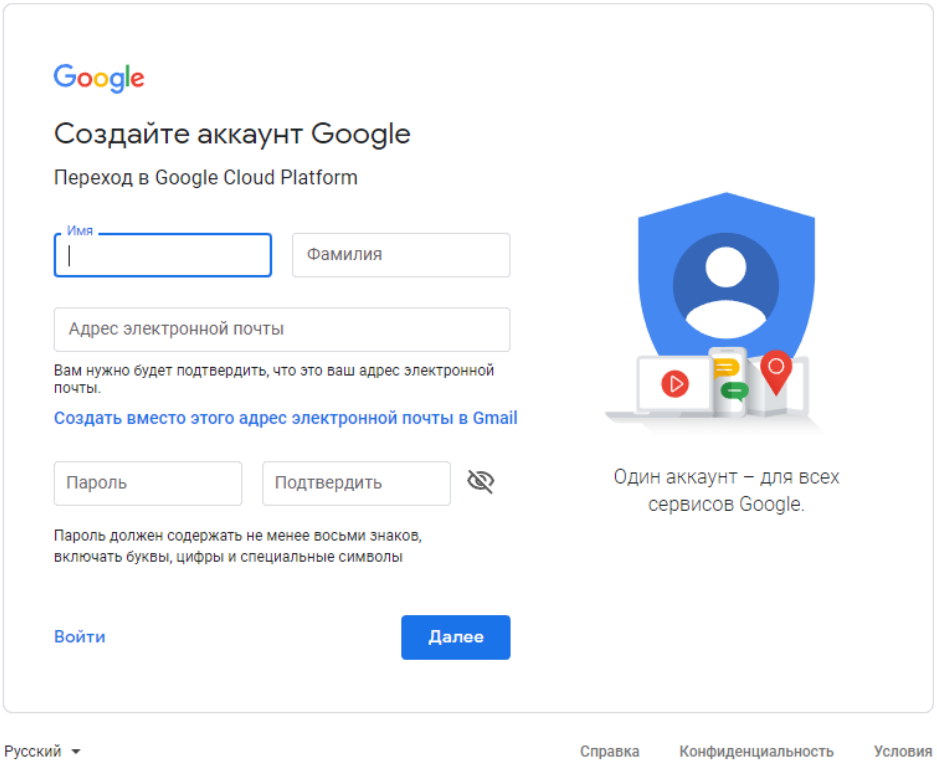

Рис 3.10 Вікно Створення акаунту GCP

Та внести необхідні данні (Рис 3.10)

Наступним кроком буде перехід на https://console.cloud.google.com

та авторизація там, за допомогою новоствореного гугл акаунту (Рис 3.11)

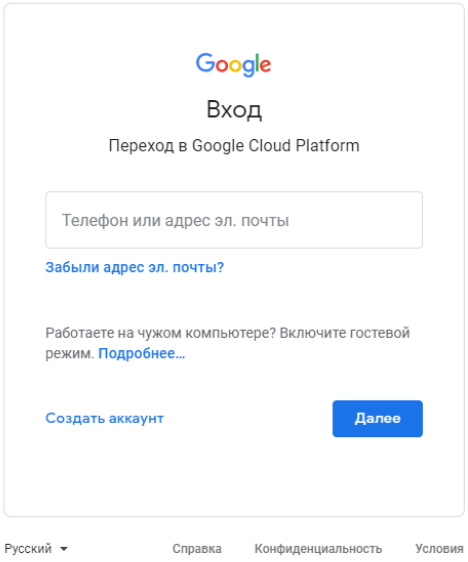

Рис 3.11 Вікно авторизації в GCP

Останнім кроком реєстрації, буде прив’язання банківського рахунку (Рис 3.12)

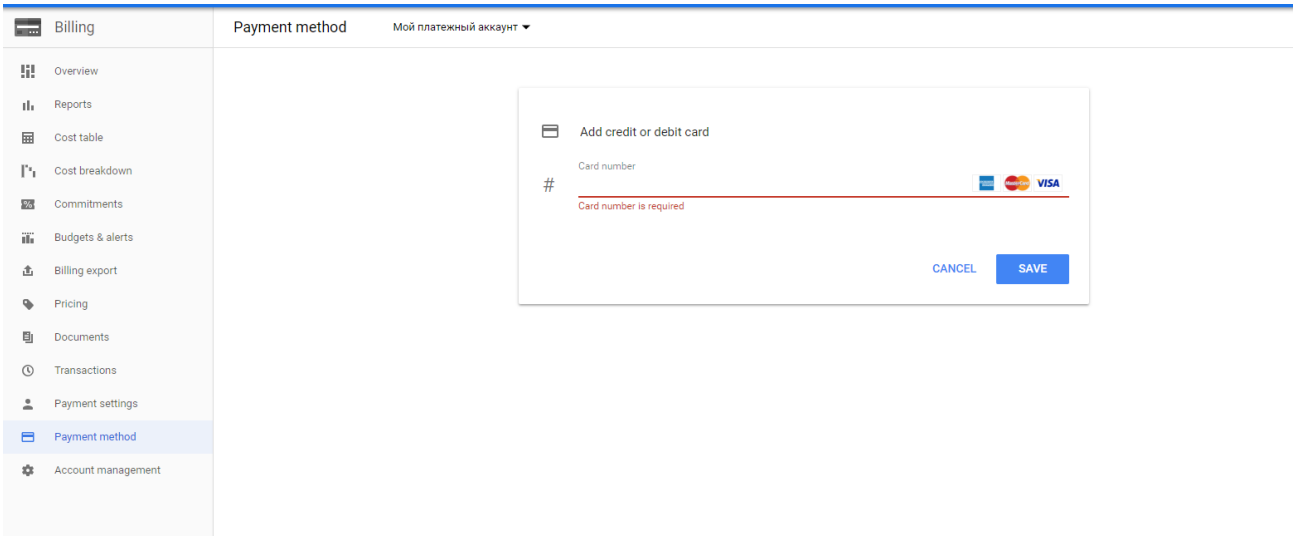

Рис 3.12 Вікно додавання кредитної карти в GCP

Компані Google зніме з вказаного рахунку 1 цент, а потім поверне. Це потрібно для ідентифікації людини і перевірки існування рахунку

Створення проекту в GCP

Перший проект (Рис 3.13) GCP створює автоматично

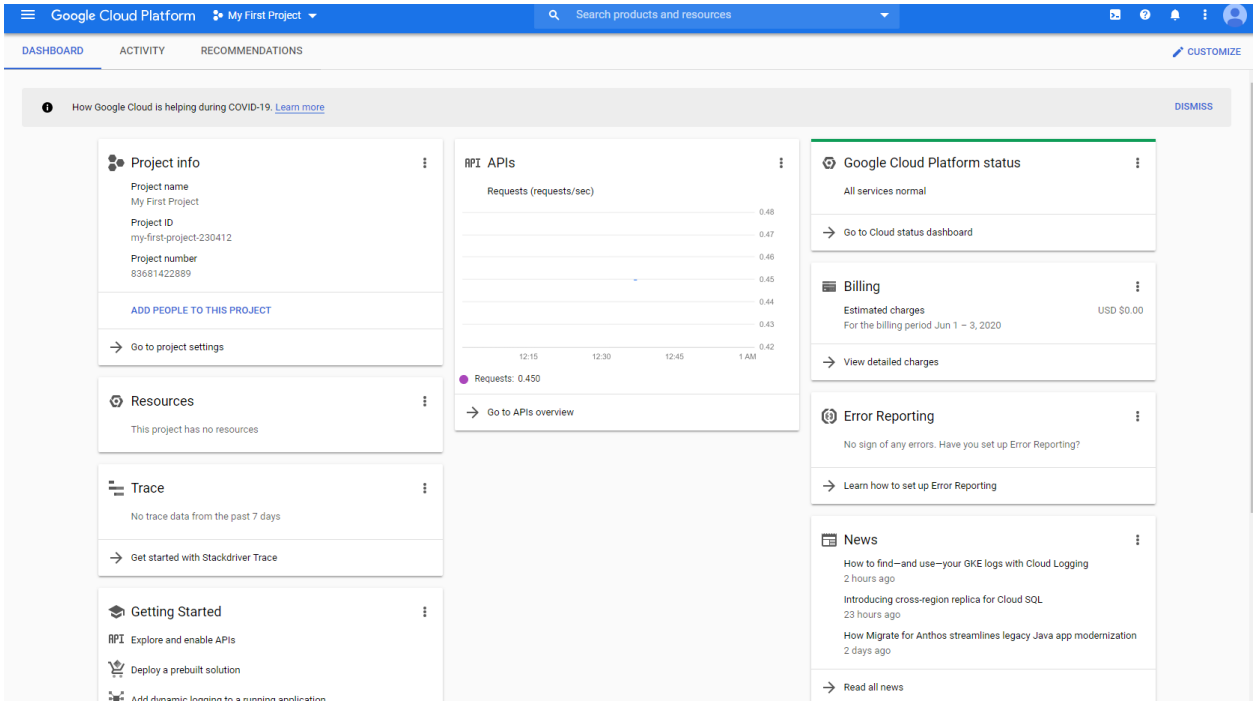

Рис 3.13 Приклад автоматично створеного проекту

Google Container registry

Потрібно ввести в поле пошукову Container Registry (Рис 3.14) і перейти на нову вкладку

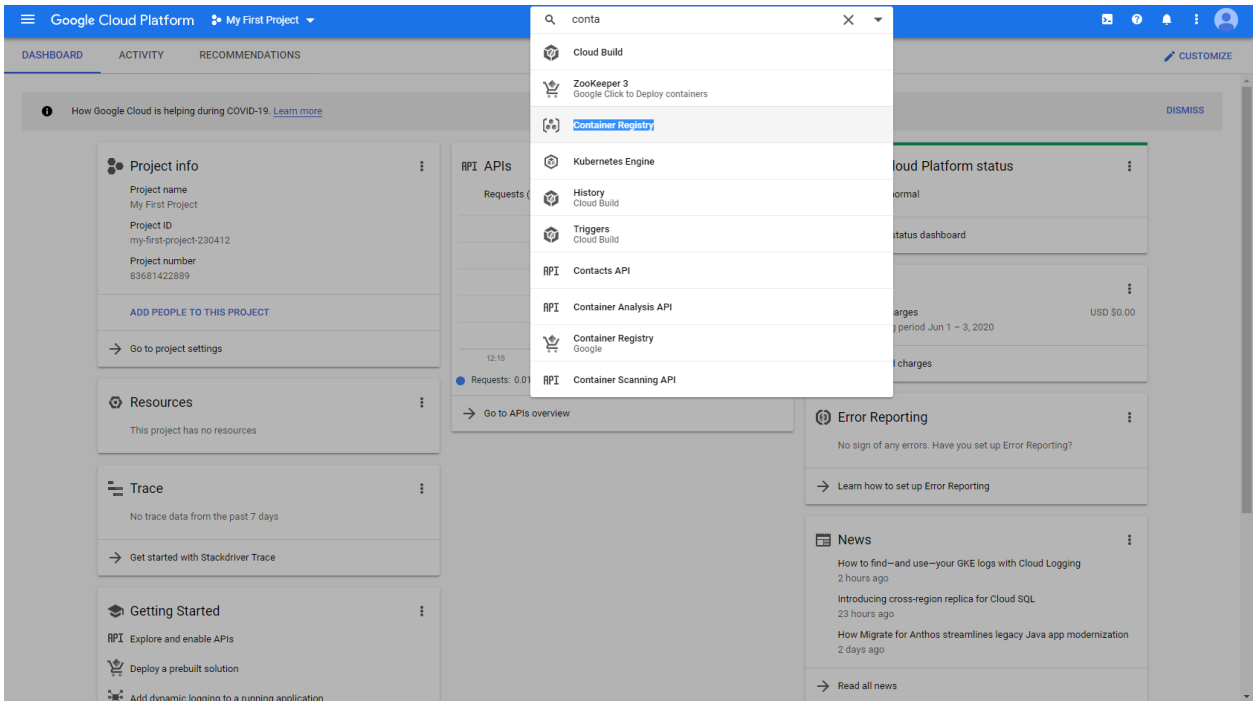

Рис 3.14 пошук GCR

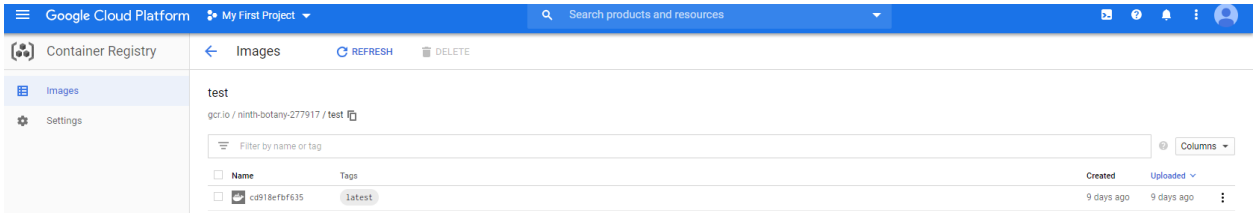

Рис 3.15 Перелік зображень в GCP (Рис 3.15)

Тут можна переконатися, що контейнер був загружений в container registry (Рис 3.15)

Cloud Run

Потрібно ввести в поле пошукову Cloud Run (Рис 3.16) і перейти на нову вкладку

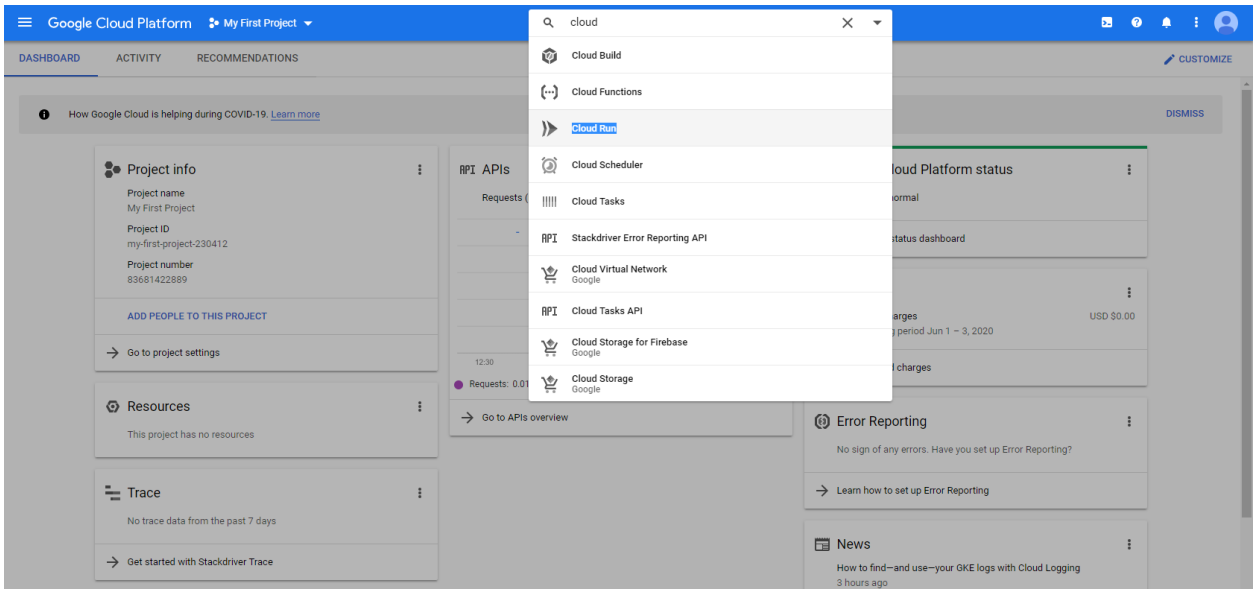

Рис 3.16 пошук Cloud Run

Після чого користувач побачить (Рис 3.17)

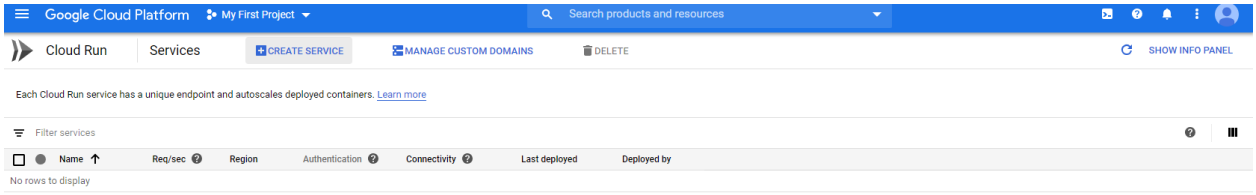

Рис 3.17 вікно Cloud Run

На цій вкладці (Рис 3.17) потрібно обрати пункт create service

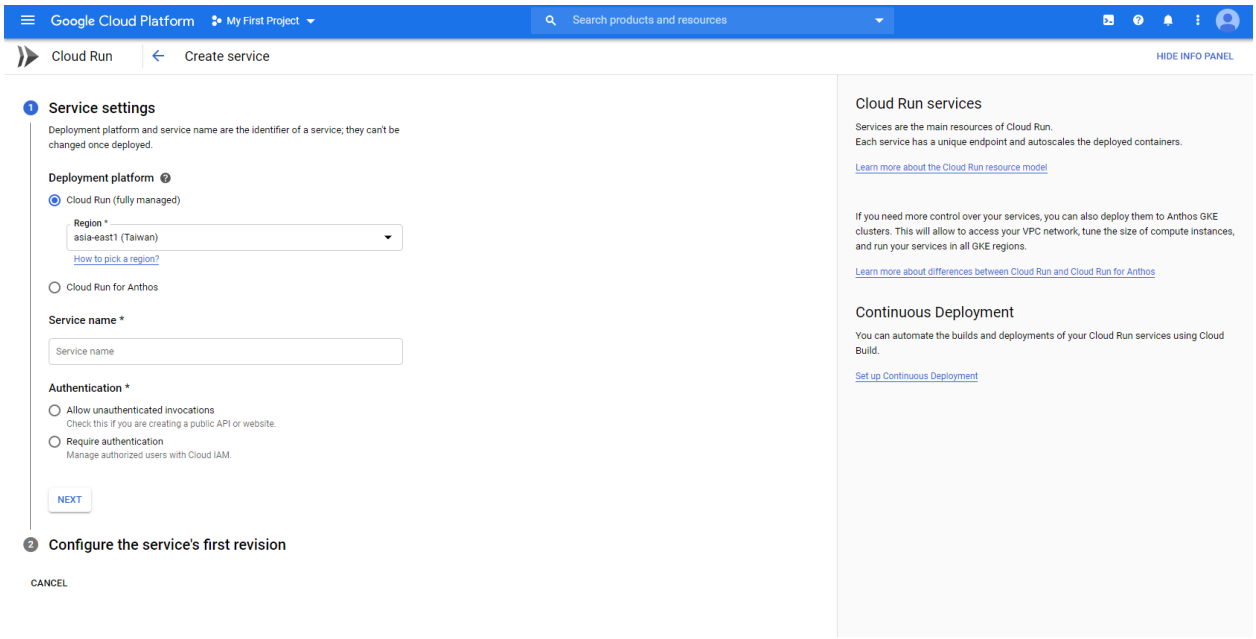

Рис 3.18 Створення сервісу в Cloud Run 1ч.

тут можна обрати назву сервісу, регіон розгортання і метод авторизації, після чого потрібно натиснути **Next** (Рис 3.18)

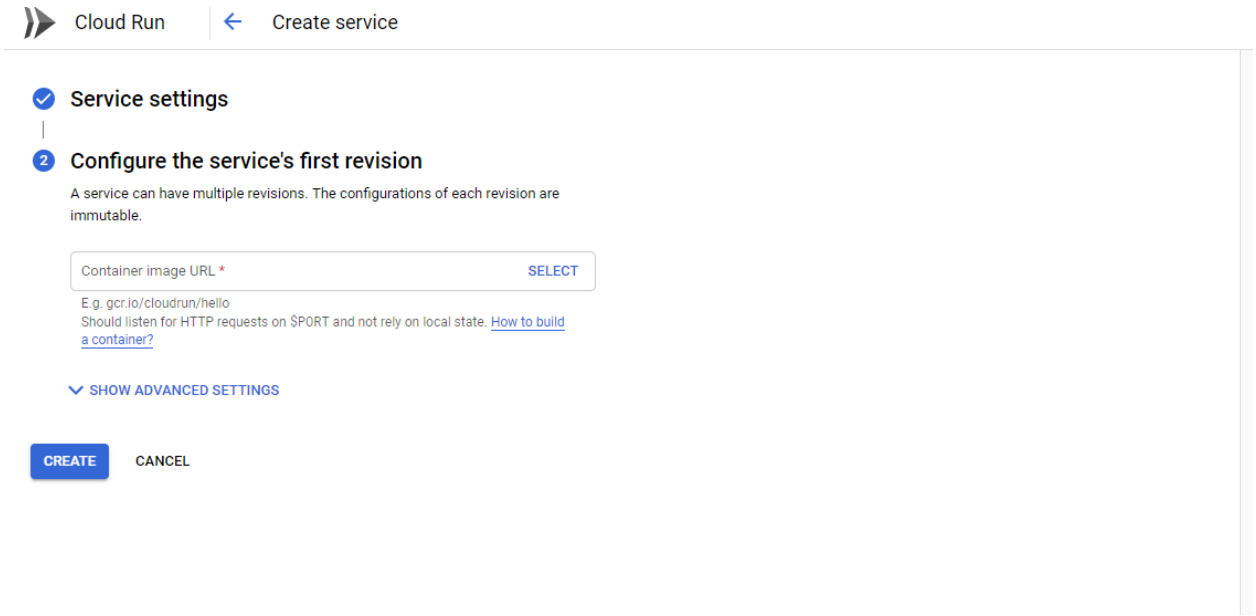

Рис 3.19 Створення сервісу в Cloud Run 2ч.

Необхідно натиснути **selec**t (Рис 3.19) і обрати із випавшого списку потрібний контейнер

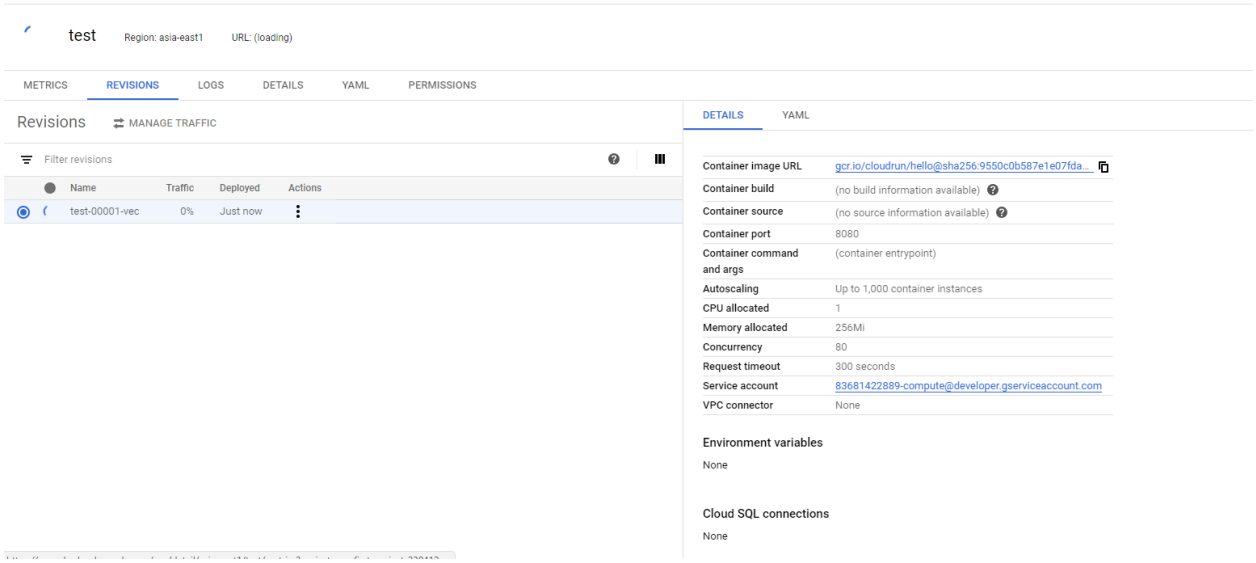

Рис 3.20 Процесс запуску контейнера в Cloud Run

Якщо користувач все виконав правильно, він побачить, що його контейнер запускається (Рис 3.20)

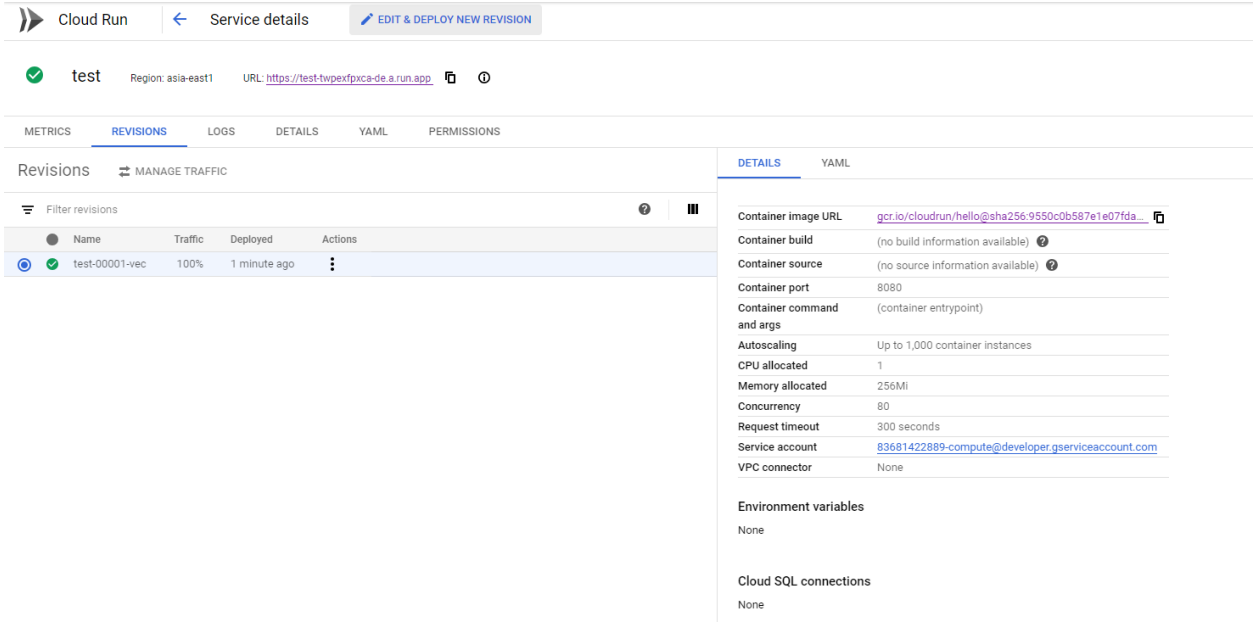

Рис 3.21 Запущений контейнер в Cloud Run

Якщо статус проекту став зеленим (Рис 3.21) - це означає, що контейнер успішно розгорнувся і запустився

Blue/Green

Для реалізації потрібно створити нову ревізію натиснувши EDIT & DEPLOY NEW PROJECT (Рис 3.21)

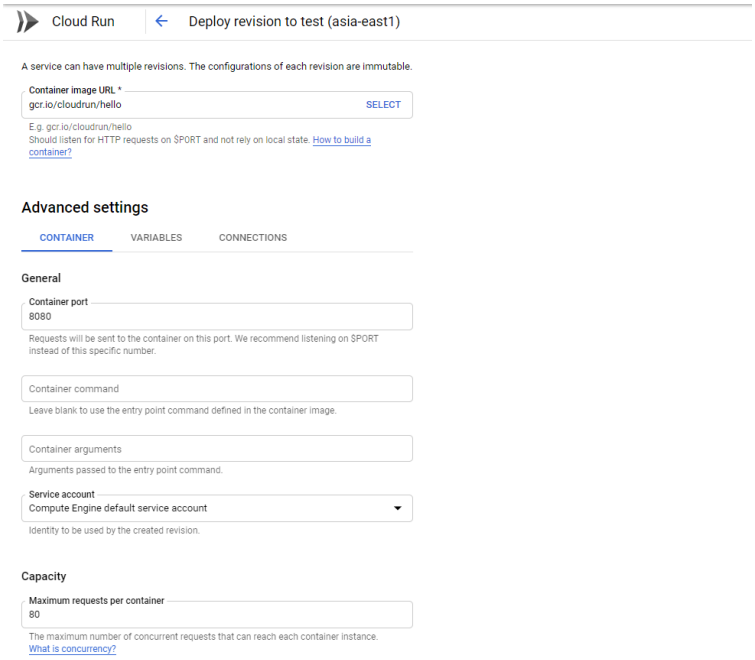

Рис 3.22 Створення нової задачі для сервіса Cloud Run 1ч.

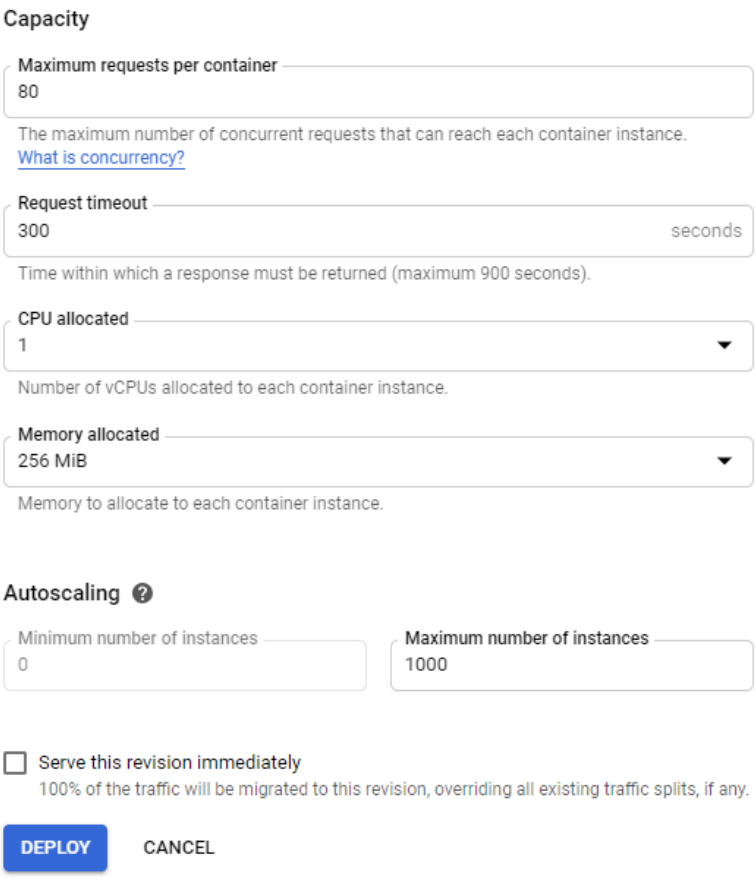

Рис 3.23 Створення нової задачі для сервіса Cloud Run 2ч.

Для успішної роботи Blue/Green, обов'язково потрібно зняти галочку з параметру **Serve this revision immediately** (Рис 3.23) - це дасть змогу контролювати трафік, що йде в контейнер

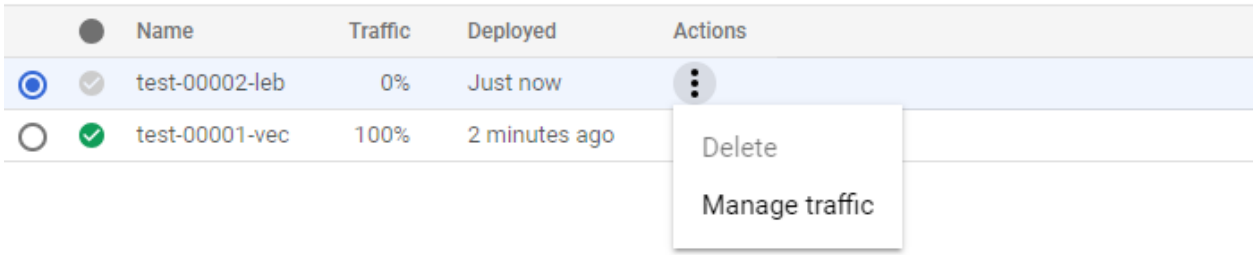

Рис 3.22 Перелік запущених контейнерів в Cloud Run та перелік можливих дій над ними

Контроль трафіку виконується за допомогою вкладки Manage traffic, в яку можна перейти обравши потрібний пункт в меню дій (Рис 3.22)

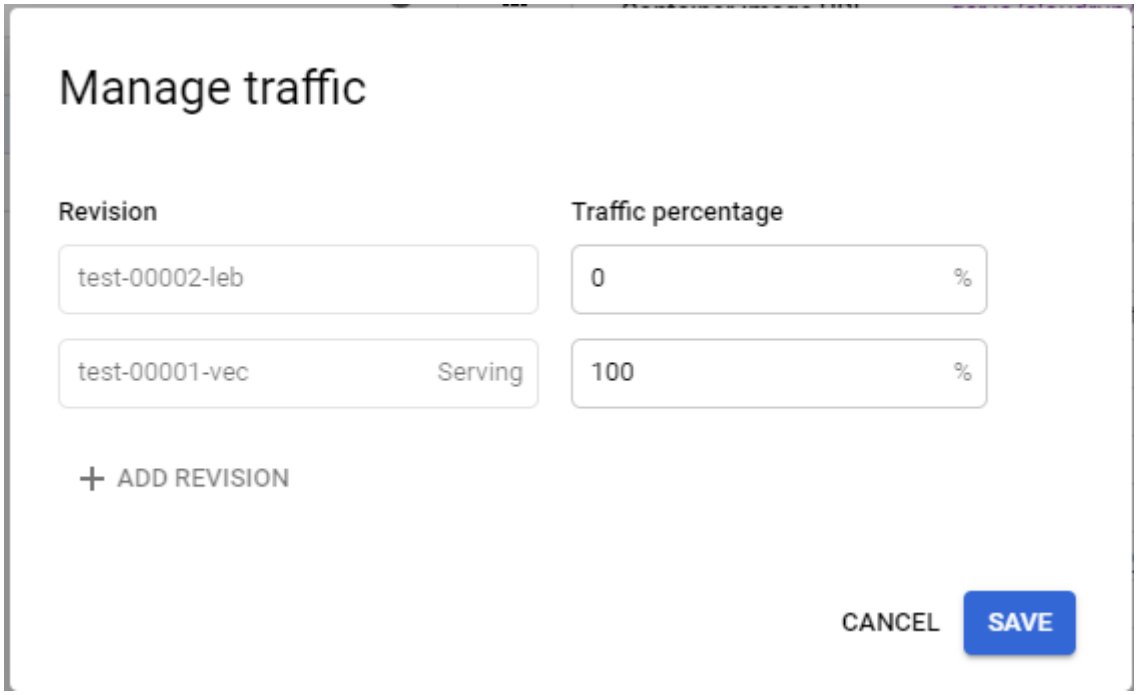

Рис 3.24 Вікно управління трафіком

В заданій вкладці (Рис 3.24) потрібно ввести процент трафіку, що буде перенаправлений до нової версії контейнеру.

Це сприяє тому, що з’являється можливість поступово, дуже пильно керувати розгортуванням контейнера та мінімізувати вірогідність похибки, вразі, якщо вона все ж таки сталася, моментально відкотити оновлення без втрати данних. Тим самим і буде реалізовановано систему blue/green.

ВИСНОВКИ

Роботу присвячено вирішенню задачі автоматизації процесу збірки тестування та розгортання додатків, в мережі клауд систем, на основі технології тестування, розгортання Blue/Green.

Було проведено теоретичний огляд та аналіз особливостей технологій Jenkins, GitLab, Travis CI, CircleCi, AWS, GCP, AZURE, Docker, VM, Blue/Green.

Підтверджено необхідність нового підходу до вирішення питання автоматизації, через неефективну роботу існуючих алгоритмів.

У якості базового алгоритму автоматизації, було обрано зв’язку Gitlab + Docker + GCP + Blue/Green через максимальну економічність та максимальну потужність. Gitlab за рахунок безкоштовної версії та виділеному серверу дає змогу використовувати автоматизацію невеликих проетків безкоштовно. Docker тому що, він створює мінімльні по обьєму зображення і здешевлює процесс роботи на сервері.GCP щерез те що, в це хмарне середовище націлене на роботу клаудів також за їх обширні можливості в використанні безкоштовноїї версії. Blue/Green через те що, ця технологія дозволить розробникам максимізувати якість розгортування, що збільшить якість кінцевого продукту. Використання запропонованого рішення дозволяє, збільшити якість кінцевого продукту та мінімізувати ціну на її роботу. Виходячи з цього можна стверджувати що всі поставлені задачі успішно виконано.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. GitLab Documentation [Електронний ресурс] // Gitlab Doc – Режим доступу до ресурсу: https://docs.gitlab.com/

2. Docker Documentation [Електронний ресурс] // Docker Doc - Режим доступу до ресурсу: https://docs.docker.com/

3. Google Cloud Platform Documentation [Електронний ресурс] // GCP Doc - Режим доступу до ресурсу: https://cloud.google.com/docs

4. BlueGreenDeployment [Електронний ресурс] // martinfowler - 2019 - Режим доступу до ресурсу: https://martinfowler.com/bliki/BlueGreenDeployment.html

5. Jenkins vs. GitLab [Електронний ресурс] // about.gitlab - Режим доступу до ресурсу: https://about.gitlab.com/devops-tools/jenkins-vs-gitlab.html.

6. Jenkins User Documentation [Електронний ресурс] // Jenkins Doc - Режим доступу до ресурсу: https://www.jenkins.io/doc/

7. Travis CI User Documentation [Електронний ресурс] // Travis Docs - Режим доступу до ресурсу: https://docs.travis-ci.com/

8. CircleCI Documentation [Електронний ресурс] // CircleCi Docs - Режим доступу до ресурсу: https://circleci.com/docs/

What’s the Diff: VMs vs [Електронний ресурс] // backblaze - 2018 - Режим доступу до ресурсу: https://www.backblaze.com/blog/vm-vs-containers/https://sibac.info/studconf/tech/xix/37782.